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

Junyang Shi, Aitian Ma, Xia Cheng, Mo Sha*, Xi Peng

***Abstract*—Today, wireless mesh networks (WMNs) are deployed globally to support various applications, such as industrial automation, military operations, and smart energy. Significant efforts have been made in the literature to facilitate their deployments and optimize their performance. However, configuring a WMN well is challenging because the network configuration is a complex process, which involves theoretical computation, simulation, and field testing, among other tasks. Our study shows that the models for network configuration prediction learned from simulations may not work well in physical networks 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 and a physical sampling method 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.**

***Index Terms*—Network Configuration, Wireless Mesh Networks, Industrial Wireless Networks, Domain Adaptation.**

I. Introduction

Recent years have witnessed the rapid deployments of wireless mesh networks (WMNs) for industrial automation [2], [3], military operations [4], and smart energy [5], 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 [6]. 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 [7], ISA [3], IEC [8], and ZigBee [9] are actively pushing the real-world implementations of WSANs for industrial automation. For example, more than 54,835 WSANs that implement the WirelessHART standard [2] have been deployed globally by Emerson Process Management to monitor and control industrial processes [10].

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: 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 in the literature to investigate the characteristics of wireless communication. 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 [11]. 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 [12], Cooja [13], OMNeT++ [14], and NS-3 [15].

However, it is still very challenging 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 [2] 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:

- This paper presents the simulation-to-reality gap in network configurations and shows that the network configuration models learned from simulations cannot always help physical networks meet performance requirements;

Junyang Shi is with Google. He contributed to this work while he was advised by Mo Sha in the Department of Computer Science at the State University of New York at Binghamton, Binghamton, NY, 13902 USA. E-mail: jshi28@binghamton.edu.

Aitian Ma, Xia Cheng, and Mo Sha are with the Knight Foundation School of Computing and Information Sciences at Florida International University, Miami, FL, 33199 USA. E-mail: {aima, xchen075, msha}@fiu.edu.

Xi Peng with the Department of Computer and Information Sciences at University of Delaware, Newark, DE, 19716 USA. E-mail: xipeng@udel.edu.

*Corresponding author.

Part of this article was published in Proceedings of the NSDI [1].

Manuscript received XX, 2022; revised XX, 2022.

- This paper presents 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, this work represents the first experimental study of the effectiveness of domain adaptation in closing the simulation-to-reality gap in network configurations;
- This paper presents an experimental method for the sampling of physical data, which measures the gap between simulation and reality in different networks. This measurement is leveraged to identify samples that need to be taken from physical networks;
- Our method has been implemented and evaluated using four simulators and a physical testbed. The evaluation has been repeated 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.

Our paper is organized as the following sections. Section II reviews the related work. Section III introduces the background of WirelessHART networks. Section IV presents our problem formulation, the simulation-to-reality gap, and our method that closes the gap. Section V shows the design of our teacher-student neural network. Section VI discusses the design of our physical data sampling method. Section VII evaluates our method. Section VIII concludes the paper.

II. 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. 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 [16], game theory [17], and meta heuristics [18]. For instance, the characteristics of low-power wireless links have been studied empirically with different platforms, under varying experimental conditions, assumptions, and scenarios [11]. 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 [19], [20]. 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 [21], [22] and reinforcement learning has been employed to enable network self-configurations [23]. The key behind the remarkable success of those data-driven methods is the capability of optimizing a huge number of free parameters [24], [25] to capture extensive uncertainties, variations, and dynamics in real-world wireless deployments. These methods do not only yield complex features, such as communication signal characteristics, channel quality, queuing state of each device, and the 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 is costly and not easily accessible (e.g., the ones in industrial facilities); 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. The 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 [26]–[29]. 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 [30].

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 target domains are associated with the same label space. It has been successfully used in computer vision [31], [32], natural language processing [33], and building occupancy estimation [34], [35]. 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 [36] or correlation distances [37]. 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 (the expectations of the source and target domain) [38], [39]. Several methods employed an adversarial loss to minimize domain shift and learn a representation that is simultaneously discriminative of source labels while not being able to distinguish between domains [40], [41]. 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.

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

III. Background of WirelessHART Networks

To meet the stringent reliability and real-time requirements of industrial applications, WirelessHART networks [2] made a set of specific design choices that distinguish themselves from traditional WSNs designed for best effort services [6]. 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 [42].

WirelessHART networks adopt the time-slotted channel hopping (TSCH) technique [43], 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 $10ms$ 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 band and performs channel hopping in every time slot to combat narrow band interference. The WirelessHART standard specifies the use of all available channels after a human operator manually blacklists noisy ones [2].

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. Emerson Process Management [44] 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 [45].

IV. 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 the simulation-to-reality gap. Finally, we introduce our deep learning based domain adaptation method, which closes the gap.

A. Experimental Setup and Data Collection

We adopt the open-source implementation of WirelessHART networks provided by Li et al. [46] and configure six data flows on our testbed, which consists of 50 TelosB motes [47]. 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 two devices to serve as access points. Figure 1 shows the device deployment on our testbed and

[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.

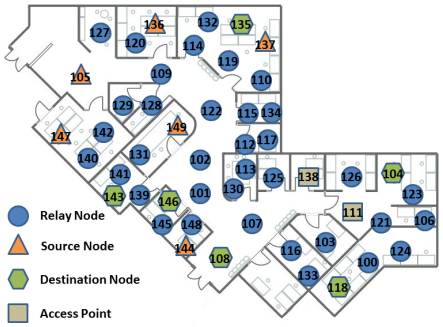


Fig. 1. Device deployment on our testbed. The device ID ranges from 100 to 149.

TABLE I
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

Table I lists the source, the destination, the data generation interval (period), and the priority of each data flow. We employ rate monotonic scheduling [48], an optimal fixed-priority policy, to generate the transmission schedule, set the data delivery deadline of each data flow to be equal to its data generation period, and configure the devices with ID 111 and 138 to serve as two access points.

We consider three configurable network parameters [49], 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[3]. We consider $R \in \{0.7, 0.71, 0.72, ..., 0.90\}$[4], $C \in \{1, 2, 3, 4, 5, 6, 7, 8\}$, and $A \in \{1, 2, 3\}$ 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 four simulators[5], feed the PRR and noise traces, the routes, and the transmission schedule collected from the physical network into the simulator, and then

[3]PDR is used to quantify the end-to-end reliability. Packet delivery ratio (PDR) is the ratio of the number of packets delivered in total to the total number of packets sent from the source node to the destination node in the network.

[4]Emerson Process Management [44] recommends using a constant value (0.6 in general or 0.7 for control and high-speed monitoring) for R [45]. We did not consider R lower than 0.7 because of the consistently low reliability we observed.

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

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 per simulation. 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 physical data collection time for 6,600 data traces from our testbed is around 4 days and the energy consumption is around 23kJ. The data gathered from the simulated network and the physical network is denoted as $\mathcal{D}^s$ and $\mathcal{D}^p$, respectively.

B. Network Configuration Prediction

The primary task in network configurations is to select the configuration (the selections of parameters R, C, A) [50], which allows the network to meet the performance requirements (L, B, 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 ($\mathbf{x}$).

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 VII-B) show that it significantly outperforms traditional optimization-based methods such as Response Surface Methodology (RSM) [51] and Kriging surrogate modeling approach [52]. The latter usually suffers the issues that include limited predictive power and being vulnerable to uneven data distribution [53].

C. Feature Selection

In addition to the features (L, B, E) that represent performance requirements, we consider nine other features, which include the received signal strength RSS [54], the link quality indicator LQI [11], the background noise G [11], 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 [55], the univariate feature selection method [56], and the recursive feature elimination method [57]) to pick the most useful features. To validate the correctness of network performance features (L, B, E) selected by Shi et al. [28], we perform a feature-selection study by evaluating different network features which might play a crucial role for the network configuration prediction. 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.

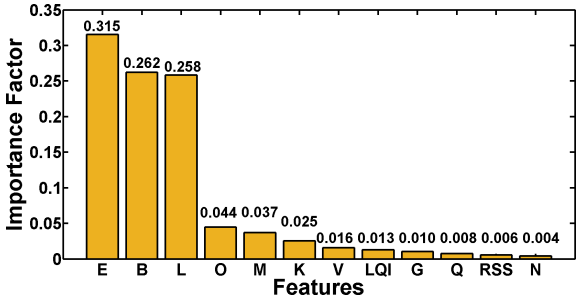


Fig. 2. Importance factors of different features when using tree-based feature selection method. Under the tree-based method, the features that are selected at the top of the trees are in general more important than the features that are selected at the end nodes of the trees, as generally, the top splits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

Therefore, we confirm that (L, B, E) are the best combination among all the feature candidates $(E, B, L, RSS, LQI, G, O, K, M, V, Q, N)$. We use (L, B, E) as input features for the models of network configuration prediction and several baselines in Section VII. Please note that the three input features are performance requirements, and our method can accept more features when needed.

D. 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 ($\mathcal{D}^s$) because of the large physical data collection overhead. Therefore we use a small physical dataset $\mathcal{D}^p$ and a large simulation dataset $\mathcal{D}^s$ 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 into training and testing dataset with a ratio of 6:4.

TABLE II
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 VII-B).

# 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

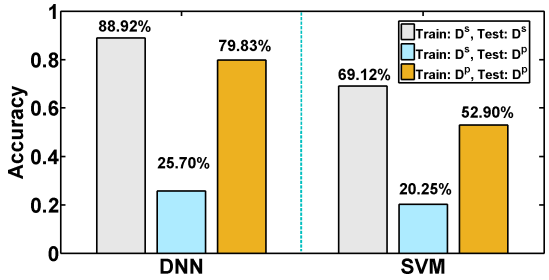


Fig. 3. Modeling accuracy when model is trained and tested on different data sets ($\mathcal{D}^s$: the simulation data produced by OMNeT++ and $\mathcal{D}^p$: the physical data). The difference between the grey bar and the blue bar indicates the simulation-to-reality gap.

The yellow bars in Figure 3 show the modeling accuracy[6], 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 (with big overhead for physical data collection) 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 IV-B 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 II 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.

[6]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 a network configuration (R, C, A), which allows the network to meet performance requirements. The accuracy is reported with 2,640 randomly chosen test data samples from the physical dataset.

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 II 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 [58], [59]. 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 VII-E). 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.

E. Close the Gap by Domain Adaptation

The observations presented in Section IV-D motivate us to explore the feasibility of using a substantial amount of inexpensive 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

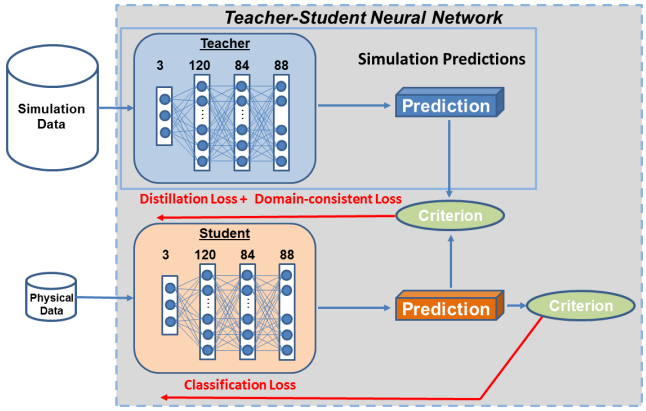


Fig. 4. Our teacher-student neural network.

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 IV-D). 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 ($f_s \rightarrow f_p$) by using a small amount of physical data (N^p). The detailed design of our teacher-student neural network will be discussed in Section V.

V. 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 IV-D) on the physical network. The teacher and student use the same architecture so that the knowledge can be transferred from the teacher network to the student network. 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.

A. 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. To keep our model light-weight, we employ Multilayer Perceptron (MLP) [60] with three layers: 120 and 84 neurons in the first two hidden layers, and 88 neurons in the output layer to represent the total 88 distinct configuration categories. Rectified linear unit (ReLU) and softmax are used to activate the hidden and output layers, respectively. The teacher's parameters (θ_1) are learned by minimizing the cross-entropy loss:

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

where $\mathcal{D}^s$ denotes the training data generated from simulations, θ_1 denotes the teacher's parameters, y denotes the one-hot label corresponding to one of the 88 considered network configurations, and $f_{\theta_1}(\mathbf{x})$ is the prediction made by the teacher. We use the Adam optimizer [61] 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.

B. 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 taught 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 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 $\mathcal{L}_{dis}$: The Distillation loss is from Knowledge distillation models. Knowledge distillation was originally proposed for model compression. A big (or ensemble) model that achieves high accuracy but requires massive computation time is used as the teacher model, and a small feasible model, called the student model, is trained to imitate the behavior of the teacher model by using the output of the teacher model, called the soft label, for computing the cross entropy loss function. This loss function allows the teacher to transfer its knowledge to the student. The soft label is a smooth probability distribution, which contains the knowledge of the teacher model, i.e. not only the correct class but also the similarity/correlation between classes. 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 [62], [63]. 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 [62] 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 [64] 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) [65]. 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 maximum number of 500 epochs have been trained on the student.

VI. Sampling from Physical Network

In this section, we first present our empirical study that quantifies the simulation-to-reality gaps under different network configurations and then introduce a physical data sampling method that leverages the gaps to identify the samples needed to be collected from the physical network. Our goal is to reduce the data collection overhead without significantly sacrificing the network configuration prediction accuracy.

A. Quantify the Simulation-to-reality Gap

Under each network configuration i, we first use the following formula to compute the performance centroid C_i [66] of the simulation data $\mathcal{D}_i^s$:

$$C_i = \frac{1}{|\mathcal{D}_i^s|} \sum_{\mathbf{x}\sim D_i^s} \mathbf{x}, \tag{7}$$

where $|\mathcal{D}^s|$ is the number of the simulation data samples under the network configuration i and $\mathbf{x}$ is the simulated network performance vector.

We then use the Mahalanobis distance [67] between each physical sample p_{ij}[7] and the centroid of the simulation data C_i under the same network configuration to represent the simulation-to-reality gap:

$$L(p_{ij}, C_i; Q) = \sqrt{(p_{ijk} - C_{ik})S^{-1}(p_{ijk} - C_{ik})^T} \tag{8}$$

where Q is a probability distribution on R^N with positive-definite covariance matrix S. C_i, p_{ij} belongs to Q, and S^{-1} is the inverse of the covariance matrix S.

Figure 5 shows the boxplot of the Mahalanobis distances between five physical data samples and the centroid of the simulation data under 88 network configurations. As Figure 5 shows, the Mahalanobis distances under different network configurations differ a lot. There exist 29 network configurations under which the median Mahalanobis distances are smaller than 4, 26 network configurations under which the median Mahalanobis distances range between 4 and 8.7, and 33 network configurations under which the median Mahalanobis distances are larger than 8.7. For example, the Mahalanobis distances between five physical data samples and the centroid of simulation data under network configuration 15 range between 0.37 and 2.34 (0.37, 0.75, 1.09, 1.72, and 2.34), the Mahalanobis distances between five physical data samples and the centroid of simulation data under network configuration 48 range between 8.61 and 8.69 (8.61, 8.61, 8.62, 8.67, and 8.69), while the Mahalanobis distances under the network configuration 76 is 8.72.

Figure 6 provides a more detailed look at the distribution of the simulation data samples and the physical data samples under network configurations 15 and 76. As Figure 6(a) shows, the simulated latency ranges between $160ms$ and $162.5ms$, the simulated battery lifetime ranges between 23.9 $days$ to 24.05 $days$, and the simulated reliability ranges between 0.97 and 1. All five network performance samples collected from the physical network fall into those ranges, which demonstrates a small discrepancy between the simulation data and the physical data under the network configuration 15. As a comparison, Figure 6(b) shows a larger discrepancy between the simulation data and the physical data under the network configuration 76 when the simulated latency values that range from $200ms$ to $500ms$ fall outside of the simulated values. Therefore, it is beneficial to collect more physical data samples where a larger simulation-to-reality gap is observed.

B. Physical Data Sampling Algorithm

As discussed in Section IV-D, sampling from a physical network consumes significant time and energy, making our solution unaffordable for some deployments. Moreover, the network must operate under all configurations to collect sufficient data to train a network configuration model, which may result in undesirable network performance. For example, the end-to-end reliability of the network is 0.26 and the end-to-end latency is $1.50s$, when it operates under Network Configuration 81. Therefore, it is important to reduce the number of physical data samples needed to train a good network configuration model. To address the issue, we develop a physical data sampling method to reduce the physical data needed for training without significantly sacrificing the network configuration prediction accuracy. Our key idea is to stop collecting physical

[7] p_{ij} is a 3-dimensional performance vector, which represents the j-th sample under the i-th network configuration

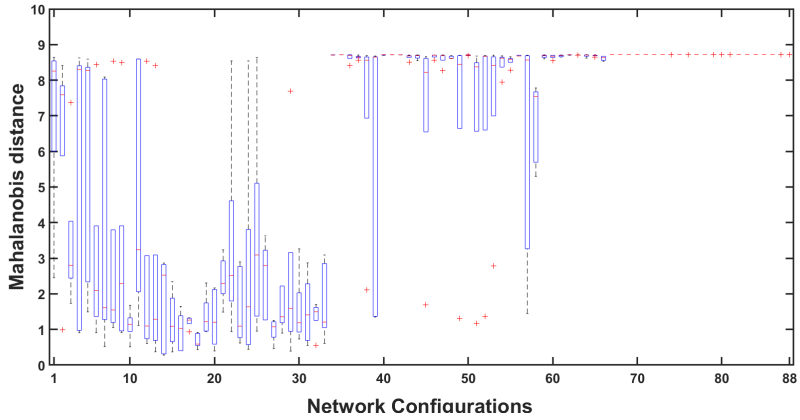
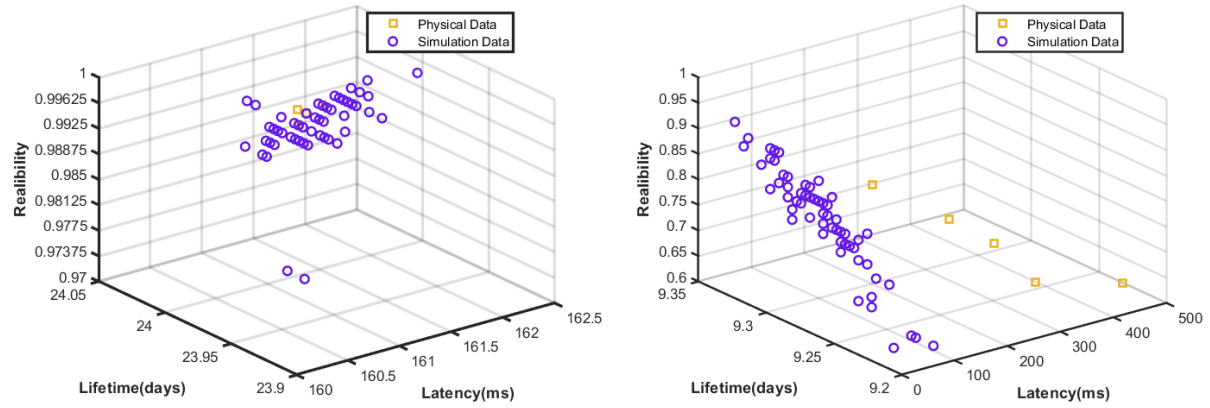


Fig. 5. Boxplot of the Mahalanobis distances between 75 simulated network performance samples and five measured network performance samples collected from the physical network under 88 network configurations. Central mark in the box indicates the median; bottom and top of the box represent the 25th percentile (q_1) and 75th percentile (q_2); crosses indicate outliers ($x > q_2 + 1.5 * (q_2 - q_1)$ or $x < q_1 - 1.5 * (q_2 - q_1)$); whiskers indicate range excluding outliers.



(a) Under the network configuration 15. (b) Under the network configuration 76.

Fig. 6. Five simulated network performance samples and 75 measured network performance samples collected from physical network.

data samples under the network configurations where either the discrepancy between the simulation data and the physical data or the discrepancy among physical data samples is small.

Algorithm 1 shows the design of our algorithm. Algorithm 1 takes the simulation data $\mathcal{D}^s$ and the maximum number of shots of data $N^{P_{MAX}}$, which can be collected from the physical network, as input, and outputs the selected physical samples $\mathcal{D}^p$. Algorithm 1 first computes the centroid of the simulation data using Eq. 7 (Line 2) and the maximum Mahalanobis distance among all simulation data samples under each network configuration (Line 3−6). Algorithm 1 decides to collect a new physical data sample under the current network configuration if either the collected samples under this configuration demonstrate a large simulation-to-reality gap or there exists a large variation between collected samples (Line 7−13). Specifically, Algorithm 1 compares the Mahalanobis distance between each physical data sample and the centroid of the simulation data against the maximum Mahalanobis distance among simulation data samples ($L(p_{ij}, C_i) \leq MAX(L_i)$). It also compares the minimum Mahalanobis distance between the current data sample and the previous data samples against the Mahalanobis distance between the current data sample and the centroid of the simulation data ($\min L(p_{i0..j-1}, p_{ij}) < |(L(p_{ij}, C_i) - MAX(L_i))|$). The algorithm leverages those comparisons to determine whether to collect more physical data under the current configuration for training.

VII. 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) [28], [29]; (iii) Fine-tuning (FT) method [68]; (iv) CCSA: Unified deep supervised domain adaptation and generalization [69]; and (v) Domain adaptive neural network (DaNN) [70], and two non-machine learning methods: (vi) RSM method [51], [71] and (vii) Kriging method [52], [72]. All methods use L, B, and E as

Algorithm 1: Physical Data Sampling Algorithm

Input : $\mathcal{D}^s$, N^p_{MAX} the maximum number of shots
Output: $\mathcal{D}^p$

for *each network configuration* i **do**
  Compute the centroid of the simulation data C_i using Eq. 7;
  for *each simulation data sample* s_{ik} *under network configuration* i **do**
    Compute $L(s_{ik}, C_i)$ between s_{ik} and C_i using Eq. 8
    Record maximum $L(s_{ik}, C_i)$ and save it to $MAX(L_i)$;
  end
  for $j = 1; j \leq N^p_{MAX}; j++$ **do**
    Measure network performance and add the sample p_{ij} to $\mathcal{D}^p$;
    Compute $L(p_{ij}, C_i)$ using Eq. 8
    if $L(p_{ij}, C_i) \leq MAX(L_i)$ *or* $minL(p_{i0..j-1}, p_{ij}) < |(L(p_{ij}, C_i) - MAX(L_i))|$ **then**
      Break;
    end
  end
end

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.

A. Experimental Setup

As presented in Section IV-A, we configure six data flows on our testbed. On each data flow, sensor data is generated by 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 $50s$. We employ the same DNN architecture for the teacher and the student in our method with independent weights (see Section V). Each neural network has 120 and 84 neurons in the first two hidden layers, and 88 neurons in the output layer. The weight α is 1, 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

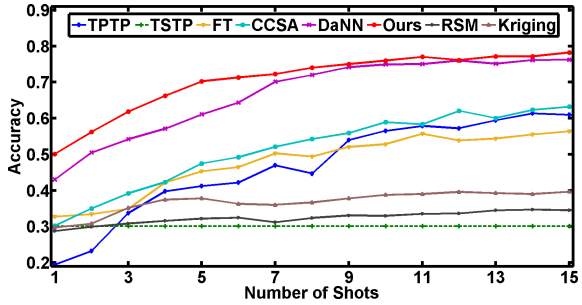


Fig. 7. Modeling accuracy of our method and baselines when different number of shots of physical data are added into simulation data (3,960 data samples in total) for training. One shot includes 88 data samples (one data sample under each network configuration).

configurations. Specifically, RSM is a black-box modeling technique and uses polynomial functions to approximate the model functions between the inputs and the outputs [51], while Kriging leverages spatial interpolation that uses complex mathematical formulas to estimate values at unknown points based on the values, which are already sampled [52]. We implement our method and baselines using python3.6 with PyTorch [73], NumPy [74], and scikit-learn [75] and run all experiments with a Dell server, Raspberry Pis and TelosB motes.

B. 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 IV-A. 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 7 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 8 plots, collecting one shot of physical data (one data sample under each of 88 network configurations) takes 1.22 hours and consumes $310.61J$ of energy. Please note that TSTP uses only the simulation data for training 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

TABLE III
FIVE EXAMPLE NETWORK CONFIGURATIONS SELECTED BY OUR METHOD AND TSTP. FIGURE 9 AND 10 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

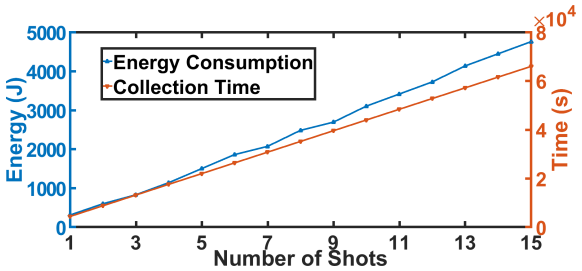


Fig. 8. Time and energy consumption to collect different number of shots of data from a physical network. Using only physical data to train the model is infeasible due to unacceptable time and energy overhead.

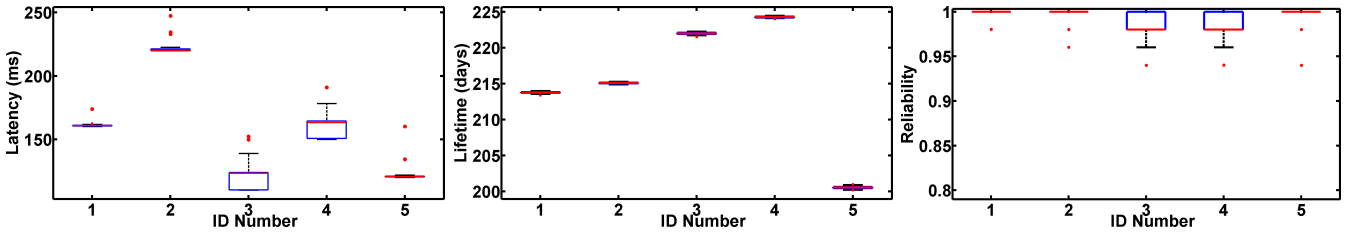
from a physical network takes a long time and consumes much energy. As Figure 8 plots, the collection of five shots of physical data takes 6.11$hours$ and consumes 1,502.88J of energy, while collecting 15 shots take 18.33$hours$ and consumes 4,758.70J of energy. The improvement on the modeling accuracy is largely shadowed by the significantly increased data collection overhead. Therefore, we use five shots in the rest of our evaluation. Figure 7 and 8 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 models 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 III lists five example network configurations selected by our method and TSTP when facing different network performance requirements. Figure 9 plots the boxplots of latency, battery lifetime[8], and reliability when employing five network configurations selected by our method. As Figure 9 shows, our method always helps the network meet the network performance requirements posed by the application (listed in Table III). For instance, the latency, battery lifetime, and reliability requirements are 170ms, 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 161.00ms, 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 165ms, 224$days$, and 95% in the fourth example (ID = 4). When employing the 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 [28], [29]

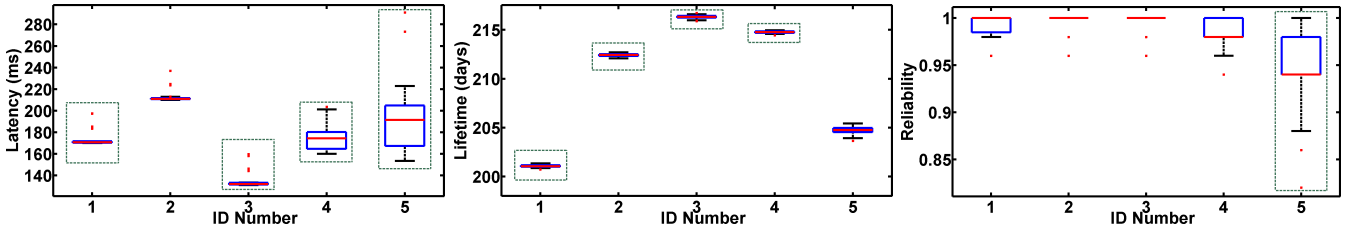
As a comparison, we also employ the network configurations selected by TSTP when facing the same network performance requirements. Table III lists the network config-

[8]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.

Fig. 9. Network performance when employing the network configurations selected by our method (listed in Table III). 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.



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

Fig. 10. Network performance when employing the network configurations selected by TSTP (listed in Table III). The dotted boxes highlight the network performance that fails to meet the requirements. Compared to Figure 9, our method always provides better network configurations than TSTR and help the network meet the application performance requirements.

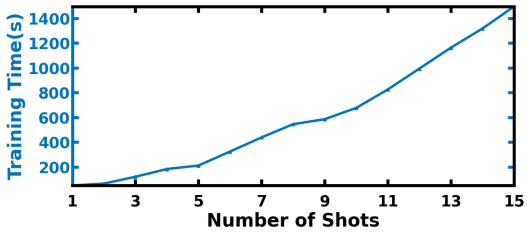


Fig. 11. Time consumed to train the model when using different amount of physical data.

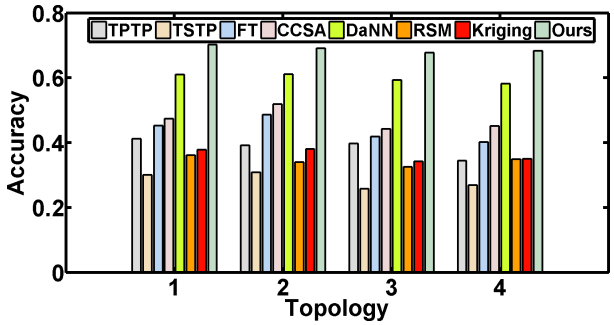


Fig. 12. Accuracy comparison among different methods with different network topologies. All methods use five shots of physical data. Topology 1 is used for Figure 7.

urations selected by TSTP and Figure 10 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 10 highlight the network performance that fails to meet the application requirements listed in Table III. For instance, the latency, battery lifetime, and reliability requirements are $130ms$, $200days$, 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 $204.74days$, and a median reliability of 94.00%, which fail to meet the latency and reliability requirements.

Finally, we measure the time consumed by our method to train a network configuration model. Figure 11 plots the time consumption when our method uses different numbers of shots of physical data for training on the computer equipped with a 2.6GHz 64-bit hexa-core CPU and 16GB memory. The time consumption increases from $49s$ to $1{,}496s$ when the physical data increases from one shot to 15 shots. The results also emphasis the importance of leveraging a small amount of physical data to train the network configuration model.

C. 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 12 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 VII-B 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 and run Jamlab [76] on them to generate controlled

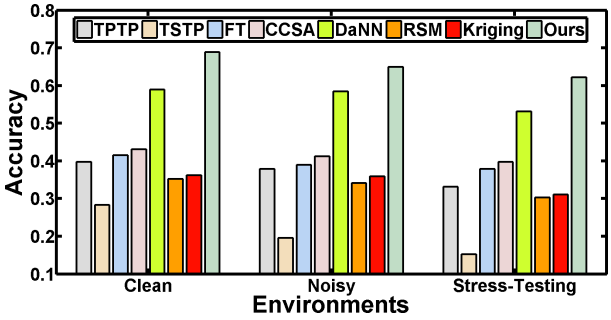
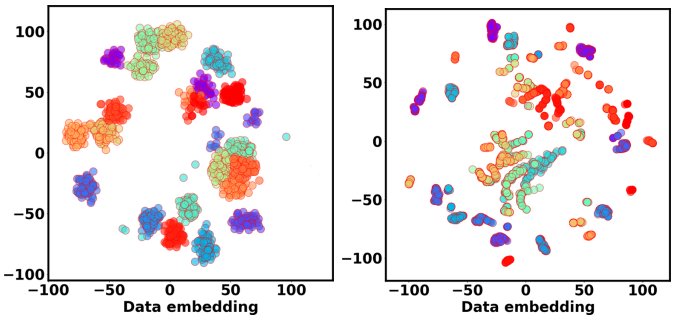


Fig. 13. Accuracy comparison among different methods under different wireless conditions.



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

Fig. 14. Data visualization provided by t-SNE [77]. Larger variations are observed in stress-testing environment, which significantly increase the difficulty on transferring knowledge learned from simulations to a physical network.

WiFi interference with various strengths to formulate different wireless conditions. 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 13 plots the modeling accuracy under three wireless conditions when employing our method and seven baselines. As Figure 13 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 the 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 [77], a dimension reduction tool for data visualization. Figure 14 shows the network performance visualization provided by t-SNE where different colors stand for different network configurations. Figure 14(a) and Figure 14(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 14(b) overlap each other. These larger variations, resulting from the interference, significantly increase the difficulty of 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 15 plots the reliability measured from the physical network and simulated by TOSSIM under four network topologies. 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 importance 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.

D. 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 1 in a clean environment. Figure 16 plots the accuracy when our method uses different combination of loss functions. As Figure 16 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).

E. Effects of Simulators and Radio Models

We further study the effects of different simulators and radio models on the performance of our method. Unit Disk Graph Medium (UDGM) [78] and Directed Graph Radio Medium (DGRM) [78] are the two most popular radio models supported by Cooja [13]. 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

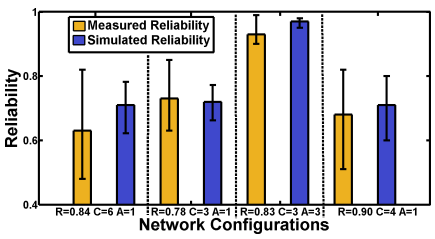


Fig. 15. Reliability measured from the physical network and simulated by TOSSIM under four network configurations.

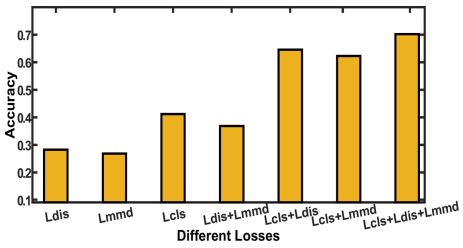


Fig. 16. Accuracy when our method uses different loss functions.

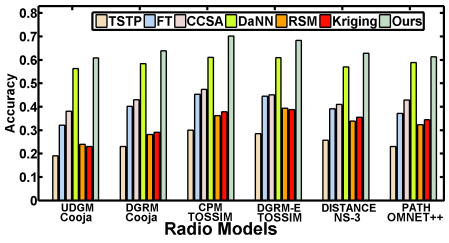
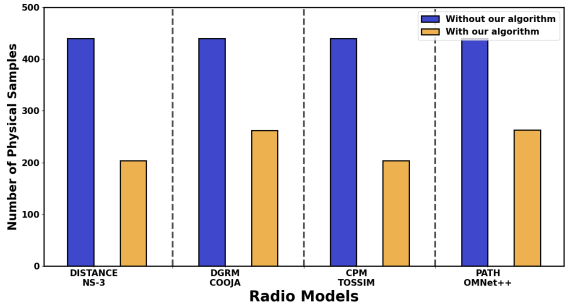


Fig. 17. Accuracy comparison when using different simulators and radio models.

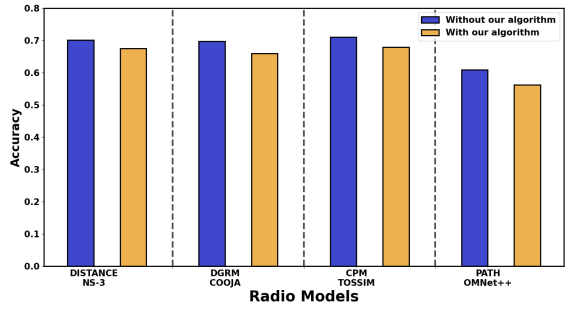
delivery ratio for each pair of devices [79]. We create DGRM-E by extending DGRM by allowing an user to specify different PRRs on different channels for each link, and then integrate it with TOSSIM. DISTANCE in NS-3 allow its user to specify the locations of all wireless devices and use the shadowing model to determine packet receptions [80]. OMNET++ allows its user 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 [81].

Figure 17 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 17 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.



(a) Number of physical data samples required for training



(b) Network configuration prediction accuracy.

Fig. 18. Performance with and without the physical data sampling algorithm

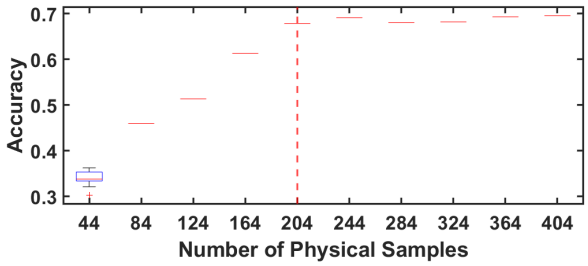


Fig. 19. Performance with different numbers of physical samples when using the TOSSIM simulator. Each experiment is repeated for 10 times with different random seeds. The physical data sampling algorithm selects 204 physical samples (marked as the red dashed line).

F. Effects of Our Physical Data Sampling Algorithm

Finally, we study the effectiveness of our physical data sampling algorithm on reducing data collection overhead. We repeat the experiments with four different simulation data

sets. Figure 18 plots the network configuration prediction accuracy and the number of samples collected from the physical network when our method enables or disables our physical data sampling algorithm. As Figure 18(a) shows, the number of physical data samples needed for training is largely reduced with the help of our physical data sampling algorithm. Instead of using all 440 physical data samples, our solution with the physical data sampling algorithm only needs 204, 262, 204, and 263 physical samples when it trains the network configuration models with the simulation data generated by the NS-3, Cooja, TOSSIM, and OMNeT++ simulators, respectively, to provide comparable prediction accuracy. As Figure 18(b) shows, the decreases on the prediction accuracy are no more than 4.7%. The results show that our physical data sampling algorithm slightly reduces the network configuration prediction accuracy provided by our method in exchange for a proportionally much larger reduction in the number of samples needed to be collected from the physical network. By further reducing the number of physical data samples needed for training, our method can be applied to large networks where collecting physical data is very time- and energy-consuming. Therefore, it is beneficial to employ our method with the physical data sampling algorithm in the deployments where the communication overhead is a major concern.

To further examine the performance of our physical data sampling algorithm, we randomly add or remove some physical samples to/from the physical sample set selected by our physical data sampling algorithm, use the new set to train the model, and then measure its prediction performance. We repeat each experiment 10 times with different random seeds. Figure 19 plots the performance when using the simulation data generated by the TOSSIM simulator. As Figure 19 shows, the median accuracy of our method is 67.88% when it uses the 204 physical samples selected by our physical data sampling algorithm for training. The median accuracy increases to 69.13% when we randomly add 40 physical samples into the target training set. The median accuracy further increases to 69.55% when we randomly add 200 physical samples into the training set. The results show that the physical samples selected by our physical data sampling algorithm is enough to provide good prediction performance and using more samples make only a marginal improvement. On the other hand, the prediction accuracy drops sharply when we randomly remove some physical samples from the selected set. For instance, the median accuracy decreases from 67.61% to 61.33% when we reduce the physical samples from 204 to 164. The median accuracy further decreases to 51.36% when we remove 40 more samples from the training set. The results show that our physical data sampling algorithm can effectively keep the important knowledge on network configuration in the training data and remove the redundancy.

VIII. Conclusions

Over the past two decades, WMNs have been widely used for industrial automation, military operations, smart energy, etc. Benefiting from years of research, WMNs work well most of the time. However, they are often difficult to configure as the WMN configuration 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 conditions. Unfortunately, our study shows that many network configurations identified in simulations do not work well 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 carefully selected from the physical network. Our experimental results show that our method consistently outperforms seven baselines and achieves high network configuration prediction accuracy.

Acknowledgment

This work was supported in part by the National Science Foundation under grant CNS-2046538 and CNS-2150010.

References

[1] J. Shi, M. Sha, and X. Peng, "Adapting Wireless Mesh Network Configuration from Simulation to Reality via Deep Learning based Domain Adaptation," in *NSDI*, 2021.
[2] "WirelessHART for Industrial Automation," HART Communication Foundation. [Online]. Available: https://fieldcommgroup.org/technologies/hart
[3] "International Society of Automation (ISA)," ISA. [Online]. Available: https://www.isa.org/
[4] "Meshdynamics," Meshdynamics. [Online]. Available: https://www.meshdynamics.com/index.html
[5] "Zigbee Smart Energy," ZigBee Alliance. [Online]. Available: https://zigbeealliance.org/zigbee_products/smart-energy-monitor-2/
[6] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems," in *ICPSs*, 2016.
[7] "HART Foundation (Now FieldComm Group)," FieldComm Group. [Online]. Available: http://www.hartcomm.org/
[8] "International Electrotechnical Commission (IEC)," International Electrotechnical Commission. [Online]. Available: https://www.iec.ch/
[9] "Zigbee Alliance," ZigBee Alliance. [Online]. Available: https://zigbeealliance.org/
[10] "WirelessHART Networks Deployed by Emerson Process Management," Emerson. [Online]. Available: https://www.emerson.com/en-us/expertise/automation/industrial-internet-things/pervasive-sensing-solutions/wireless-technology
[11] N. Baccour, A. Koubâa, L. Mottola, M. A. Zúñiga, H. Youssef, C. A. Boano, and M. Alves, "Radio Link Quality Estimation in Wireless Sensor Networks: A Survey," *ACM TOSN*, vol. 8, no. 4, 2012.
[12] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *Sensys*, 2003.
[13] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with Cooja," in *LCN*, 2006.
[14] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," in *ICST*, 2008.
[15] "NS-3 Network Simulator," NS-3. [Online]. Available: https://www.nsnam.org/

[16] Z. Q. Luo and W. Yu, "An Introduction to Convex Optimization for Communications and Signal Processing," *IEEE J. Sel. Areas Commun*, vol. 24, no. 8, pp. 1426–1438, 2006.

[17] E. Altman, T.Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynterc, "A Survey on Networking Games in Telecommunications," *Computers and Operations Research*, vol. 33, no. 2, pp. 286–311, 2006.

[18] M. G. Resende and P. Pardalos, *Handbook of Optimization in Telecommunications*. Springer, 2006.

[19] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "PTunes: Runtime Parameter Adaptation for Low-Power MAC Protocols," in *IPSN*, 2012.

[20] W. Dong, C. Chen, X. Liu, Y. He, Y. Liu, J. Bu, and X. Xu, "Dynamic Packet Length Control in Wireless Sensor Networks," *IEEE TWC*, vol. 13, no. 3, pp. 1172–1181, 2014.

[21] Q. Mao, F. Hu, and Q. Hao, "Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey," *IEEE Commun. Surv. Tutor.*, vol. 20, no. 4, pp. 2595–2621, 2018.

[22] C. Zhang, P. Patras, and H. Haddadi, "Deep Learning in Mobile and Wireless Networking: A Survey," *IEEE Commun. Surv. Tutor.*, vol. 21, no. 3, pp. 2224–2287, 2019.

[23] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Commun. Surv. Tutor.*, vol. 21, no. 4, pp. 3133–3174, 2019.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[25] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely Connected Convolutional Networks," in *CVPR*, 2017.

[26] S. Tobiyama, B. Hu, K. Kamiya, and K. Takahashi, "Large-Scale Network-Traffic-Identification Method with Domain Adaptation," in *ACM WWW*, 2020.

[27] M. Bartulovic, J. Jiang, S. Balakrishnan, V. Sekar, and B. Sinopoli, "Biases in Data-Driven Networking, and What to Do About Them," in *HotNets*, 2017.

[28] J. Shi and M. Sha, "Parameter Self-Configuration and Self-Adaptation in Industrial Wireless Sensor-Actuator Networks," in *INFOCOM*, 2019.

[29] ——, "Parameter Self-Adaptation for Industrial Wireless Sensor-Actuator Networks," *ACM TOIT*, vol. 20, no. 3, 2020.

[30] J. Shi, M. Sha, and X. Peng, "Adapting wireless mesh network configuration from simulation to reality via deep learning based domain adaptation," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 887–901.

[31] M. Wang and W. Deng, "Deep Visual Domain Adaptation: A Survey," *Neurocomputing*, vol. 312, no. 27, pp. 135–153, 2018.

[32] V. M. Patel, R. Gopalan, R. Li, and R. Chellappa, "Visual Domain Adaptation: A Survey of Recent Advances," *IEEE Signal Processing Magazine*, vol. 32, no. 3, pp. 53–69, 2015.

[33] D. W. Otter, J. R. Medina, and J. K. Kalita, "A Survey of the Usages of Deep Learning for Natural Language Processing," *IEEE TNNLS*, vol. Early Access, 2020.

[34] I. B. Arief-Ang, F. D. Salim, and M. Hamilton, "DA-HOC: Semi-Supervised Domain Adaptation for Room Occupancy Prediction using CO2 Sensor Data," in *ACM BuildSys*, 2017.

[35] T. Zhang and O. Ardakanian, "A Domain Adaptation Technique for Fine-Grained Occupancy Estimation in Commercial Buildings," in *ACM/IEEE IoTDI*, 2019.

[36] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain Separation Networks," in *NIPS*. Curran Associates Inc., 2016.

[37] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, and V. Lempitsky, "Domain-Adversarial Training of Neural Networks," *JMLR*, vol. 17, no. 59, pp. 1–35, 2016.

[38] A. Gretton, A. Smola, J. Huang, M. Schmittfull, K. Borgwardt, and B. Schölkopf, *Covariate Shift and Local Learning by Distribution Matching*. MIT Press, 2009, pp. 131–160.

[39] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep Domain Confusion: Maximizing for Domain Invariance," *ArXiv*, vol. abs/1412.3474, 2014.

[40] Y. Ganin and V. Lempitsky, "Unsupervised Domain Adaptation by Backpropagation," in *ICML*, 2015.

[41] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial Feature Learning," *CoRR*, vol. abs/1605.09782, 2017.

[42] "IEEE 802.15.4e WPAN Task Group." [Online]. Available: http://www.ieee802.org/15/pub/TG4e.html

[43] "IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH)," IEEE. [Online]. Available: https://tools.ietf.org/html/rfc7554

[44] "Emerson Process Management." [Online]. Available: https://www.emerson.com/en-us/automation-solutions

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

[46] "WCPS Simulator," Washington University in St. Louis. [Online]. Available: http://wsn.cse.wustl.edu/index.php/WCPS:_Wireless_Cyber-Physical_Simulator

[47] "TelosB Datasheet Provided by Memsic Incorporation." [Online]. Available: https://www.willow.co.uk/TelosB_Datasheet.pdf

[48] J. Liu, *Real-Time Systems*, 1st ed. USA: Prentice Hall PTR, 2000.

[49] J. Shi and M. Sha, "ParameterSelf-Configuration and Self-Adaptation in Industrial WirelessSensor-Actuator Networks," in *INFOCOM*, 2019.

[50] D. Gunatilaka, M. Sha, and C. Lu, "Impacts of channel selection on industrial wireless sensor-actuator networks," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[51] M. A. Bezerra, R. E. Santelli, E. P. Oliveira, L. S. Villar, and L. A. Escaleira, "Response Surface Methodology (RSM) as a Tool for Optimization in Analytical Chemistry," *Talanta*, vol. 76, no. 5, pp. 965 – 977, 2008.

[52] T. Simpson, T. Mauery, J. Korte, and F. Mistree, "Kriging Models for Global Approximation in Simulation-Based Multidisciplinary Design Optimization," in *AIAA Journal*, 2001.

[53] C. Caruso and F. Quarta, "Interpolation Methods Comparison," *Comput. Math. with Appl.*, vol. 35, no. 12, pp. 109 – 126, 1998.

[54] K. Benkic, M. Malajner, P. Planinsic, and Z. Cucej, "Using RSSI value for distance estimation in wireless sensor networks based on ZigBee," in *IWSSIP*, 2008.

[55] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding Variable Importances in Forests of Randomized Trees," in *NeurIPS*, 2013.

[56] A. Jovic, K. Brkić, and N. Bogunović, "A Review of Feature Selection Methods with Applications," in *MIPRO*, 2015.

[57] X. Chen and J. cheol Jeong, "Enhanced recursive feature elimination," in *Sixth ICMLA*, 2007.

[58] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. P. Sampedro, K. Konolige, S. Levine, and V. Vanhoucke, "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping," in *ICRA*, 2018.

[59] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from Simulated and Unsupervised Images through Adversarial Training," in *CVPR*, 2017.

[60] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer Perceptron and Neural Networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, p. 579–588, 2009.

[61] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *ICLR*, 2014.

[62] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," in *NIPS*, 2015.

[63] T. Asami, R. Masumura, Y. Yamaguchi, H. Masataki, and Y. Aono, "Domain Adaptation of DNN Acoustic Models Using Knowledge Distillation," in *ICASSP*, 2017.

[64] G. K. Dziugaite, D. M. Roy, and Z. Ghahramani, "Training Generative Neural Networks via Maximum Mean Discrepancy Optimization," in *UAI*, 2015.

[65] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain Adaptation via Transfer Component Analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011.

[66] E.-H. S. Han and G. Karypis, "Centroid-based document classification: Analysis and experimental results," in *ECML PKDD*. Springer, 2000, pp. 424–431.

[67] Wikipedia, "Mahalanobis distance." [Online]. Available: https://en.wikipedia.org/wiki/Euclidean_distance

[68] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, "A Comparative Study of Fine-tuning Deep Learning Models for Plant Disease Identification," *Comput Electron Agric*, vol. 161, pp. 272 – 279, 2019.

[69] S. Motiian, M. Piccirilli, D. A. Adjeroh, and G. Doretto, "Unified Deep Supervised Domain Adaptation and Generalization," in *ICCV*, 2017.

[70] M. Ghifary, W. B. Kleijn, and M. Zhang, "Domain Adaptive Neural Networks for Object Recognition," in *PRICAI*, 2014.

[71] K. K. Vadde, V. R. Syrotiuk, and D. C. Montgomery, "Optimizing Protocol Interaction Using Response Surface Methodology," *IEEE TMC*, vol. 5, no. 6, pp. 627–639, 2006.

[72] G. Boccolini, G. Hernández-Peñaloza, and B. Beferull-Lozano, "Wireless Sensor Network for Spectrum Cartography based on Kriging Interpolation," in *PIMRC*, 2012.

[73] "PyTorch: An Open Source Machine Learning Framework." [Online]. Available: https://pytorch.org

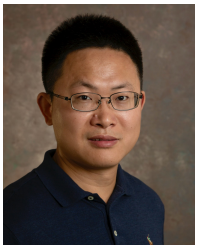
[74] "NumPy: The Fundamental Package for Scientific Computing with Python." [Online]. Available: https://numpy.org/
[75] "Scikit-learn: Simple and Efficient Tools for Predictive Data Analysis with Python." [Online]. Available: https://scikit-learn.org
[76] C. A. Boano, T. Voigt, C. Noda, K. Romer, and M. Zuniga, "JamLab: Augmenting Sensornet Testbeds with Realistic and Controlled Interference Generation," in *IPSN*, 2011.
[77] L. Van der Maaten and G. Hinton, "Visualizing Data using t-SNE ," *JMLR*, vol. 9, pp. 2579–2605, 2008.
[78] T. 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.
[79] H. Lee, A. Cerpa, and P. Levis, "Improving Wireless Simulation through Noise Modeling," in *IPSN*, 2007.
[80] "NS-3 Shadowing Model." [Online]. Available: https://www.nsnam.org/docs/release/3.10/doxygen/classns3_1_1_shadowing_loss_model.html
[81] "OMNeT++ INET Framework and Transmission Medium," OMNeT++ INET. [Online]. Available: https://inet.omnetpp.org/docs/users-guide/ch-transmission-medium.html

Junyang Shi is a software engineer at Google. He received his Ph.D. in Computer Science from State University of New York at Binghamton in 2021 and his B.S. degree in Electrical and Electronic Engineering from the Huazhong University of Science and Technology in 2016. His research focuses on industrial wireless networks and Internet of Things.

Aitian Ma is a Ph.D. student in the Knight Foundation School of Computing and Information Sciences at Florida International University. His research focuses on industrial wireless networks.

Xia Cheng is a Ph.D. student in the Knight Foundation School of Computing and Information Sciences at Florida International University. He received a M.Eng. degree from Tsinghua University in 2011 and a B.Eng. degree in Automation Engineering from Tsinghua University in 2006. His research focuses on industrial wireless networks and network security.

Mo Sha is an Associate Professor in the Knight Foundation School of Computing and Information Sciences at Florida International University (FIU). Before joining FIU, he was an Assistant Professor in the Department of Computer Science at State University of New York at Binghamton. His research interests include wireless networking, Internet of Things, applied machine learning, network security, and cyber-physical systems. He published more than 60 research papers, served on the technical program committees of 21 premier conferences, and reviewed paper for 26 journals. He received the NSF CAREER award in 2021 and the NSF CRII award in 2017. He received his Ph.D. degree in Computer Science from Washington University in St. Louis in 2014, his M.Phil. degree from City University of Hong Kong in 2009, and his B.Eng. degree from Beihang University in 2007. He is a senior and lifetime member of ACM and a member of Sigma Xi.



Xi Peng is an Assistant Professor in Department of Computer & Information Sciences at University of Delaware. His research interests include Machine Learning, Deep Learning, and Computer Vision.