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
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 iden-
tify 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\(^3\). Under TSCH, time is divided into 10\(\text{ms}\) 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

\[^3\]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.

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.

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 \(R \in \{0.7,0.71,0.72,\ldots,0.9\}\), \(C \in \{1,2,3,4,5,6,7,8\}\), and

\[^3\]Emerson Process Management \[22\] recommends using a constant value
Table 1: Data flows.

<table>
<thead>
<tr>
<th>Flow ID</th>
<th>Source</th>
<th>Destination</th>
<th>Period (ms)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>147</td>
<td>146</td>
<td>500</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>144</td>
<td>143</td>
<td>500</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>105</td>
<td>104</td>
<td>500</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>149</td>
<td>118</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>136</td>
<td>135</td>
<td>1000</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>137</td>
<td>108</td>
<td>1000</td>
<td>6</td>
</tr>
</tbody>
</table>

$A \in \{1, 2, 3\}$ 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\(^1\), 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 $(L, B, E$ values) is computed in every 50s. 75 network performance traces are collected under each network configuration. In total, we collect 6,600 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 $D^f$ and $D^p$, 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 $x = \text{concatenation}(L, B, E)$ denote the given network performance requirements and $y = \text{concatenation}(R, C, A)$ denote the configuration, which allows the network to meet performance requirements. The goal is to learn a nonlinear map $f_0: \mathbb{R}^n \rightarrow \mathbb{R}^m$. Based on the specific application, the user can set the performance requirements ($x$). The input features in $x$ are selected by the feature selection study in Section 4.3.

We use $\theta$ to denote the model parameters that are learned from data in a data-driven manner. Given the fact that the network configuration values ($y$) can be discretized without losing the generality, we further restrict $f_0$ as a discriminative model to solve a classification problem: an application can set its performance requirements ($x$), and the classifier ($f_0$) will predict the network configuration ($y$) 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 $G$, 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

---

\(^1\)We repeat our experiments using four simulators: TOSSIM, Cooja, OMNeT++, and NS-3.
Our goal is to learn a classifier to predict network configurations, no matter how large the data volume is, may not generalize well to a physical network.

### 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 (Dp) or the simulation data (Ds). Instead, we propose to use both Ds and Dp to learn the model as explained in the next section. **Using only physical data (Dp):** This would result in significant time and energy consumption due to the costly data collection process. We first leverage the physical data (Dp) 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 (Dp) into the [0, 1] range and split it randomly for training and testing. The yellow bars in Figure 3 show the 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 (Dp) that increases from 88 traces to 3,960 traces. However, the time spent on collecting the training data (Dp) increases from 1.22 hours to 55.00 hours. Moreover, the energy consumed by each field device for data collections on average increases from 310.61 J to 13,974.26 J, which represents 0.73% and 32.73% of its total energy capacity.

**Using only simulation data (Ds):** 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 27.41 s to 1,231.40 s and no energy is consumed by any field devices. However, a classifier that is trained based on the simulation data (Ds) 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 (Ds) 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 (Dp) 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 (Ds) 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-
first gather to address the domain discrepancy issue. Specifically, we solving a classification problem to using domain adaptation prediction. To this end, our objective narrows down from physical data to train the model for network configuration sive simulation data together with a small amount of costly domain) and then acquire domain). The detailed design of our teacher-student neural net-
utation data for training and the training data (Np) consists of a total number of Np 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 lay-
ers, respectively. The teacher’s parameters (θt) are learned by minimizing the cross-entropy loss:
\[
\mathcal{L}(\theta_t) = - \sum_{x \sim D^p} y \log(f_{\theta_t}(x)),
\]
where \(D^p\) denotes the training data generated from simulations, \(\theta_t\) denotes the teacher’s parameters, \(y\) denotes the one-
hot label, and \(f_{\theta_t}(x)\) 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 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) ac-

### 5.1 Teacher Neural Network

The teacher takes advantage of the large amount of simulation data for training and the training data (\(D^p\)) consists of a total number of \(N^p\) 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 (\(\theta_t\)) are learned by minimizing the cross-entropy loss:

\[
\mathcal{L}(\theta_t) = - \sum_{x \sim D^p} y \log(f_{\theta_t}(x)),
\]

where \(D^p\) denotes the training data generated from simulations, \(\theta_t\) denotes the teacher’s parameters, \(y\) denotes the one-
hot label, and \(f_{\theta_t}(x)\) 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 (\(N^s < N^p\)). To achieve

<table>
<thead>
<tr>
<th># of Data Samples Used for Training</th>
<th>From a Physical Network (Train: (D^p), Test: (D^p))</th>
<th>From Simulations (Train: (D^p), Test: (D^p))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy (%)</td>
<td>Collection Time (s)</td>
</tr>
<tr>
<td>88</td>
<td>19.39</td>
<td>4.40 \times 10^3</td>
</tr>
<tr>
<td>528</td>
<td>42.16</td>
<td>2.64 \times 10^4</td>
</tr>
<tr>
<td>968</td>
<td>57.92</td>
<td>4.84 \times 10^4</td>
</tr>
<tr>
<td>2,024</td>
<td>67.68</td>
<td>1.01 \times 10^5</td>
</tr>
<tr>
<td>3,080</td>
<td>78.82</td>
<td>1.54 \times 10^5</td>
</tr>
<tr>
<td>3,960</td>
<td>79.83</td>
<td>1.98 \times 10^5</td>
</tr>
</tbody>
</table>

Figure 4: Our teacher-student neural network.
employing the cross-entropy loss:

\[ \mathcal{L}_\text{cls} = \mathcal{L}_\text{cls} + \alpha \mathcal{L}_\text{dis} + \beta \mathcal{L}_\text{mmd} \]  

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

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

\[ \mathcal{L}_\text{cls} = - \mathbb{E}_{x \sim \mathcal{D}^p} y \log(f_{\theta_2}(x)), \]

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

**Distillation loss** \( \mathcal{L}_\text{dis} \): 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 \( \mathcal{L}_\text{dis} \) with soft labels, we use the following formula:

\[ \mathcal{L}_\text{dis} = - \mathbb{E}_{x \sim \mathcal{D}^p} \mathbf{q} \log(f_{\theta_2}(x)), \]

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

\[ \mathbf{q} = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \]

where \( T \) is the temperature [34] and \( z_i \) is the pre-softmax output of the teacher. When \( T \) increases, the soft label \( \mathbf{q} \) 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** \( \mathcal{L}_\text{mmd} \): 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:

\[ \mathcal{L}_\text{mmd} = \left\| \mathbb{E}_{x \sim \mathcal{D}_s} f_{\theta_1}(x) - \mathbb{E}_{x \sim \mathcal{D}_p} f_{\theta_2}(x) \right\| \]

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.

### 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 (TTPP); (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) Krigma 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Physical Data</td>
<td>Simulation Data</td>
</tr>
<tr>
<td>TSTP</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>TTPP</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>FT</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>CCSA</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>DaNN</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>RSM</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Ours</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>
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 5). Each neural network has 120 and 84 neurons in the first two hidden layers, and 88 neurons in the output layer. The weight $\beta$ 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 310.61J 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 6.11hours and consumes 1,502.88J of energy, while collecting 15 shots take 18.33hours 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 5 and 6 also show that only using physical data to train the model is inefficient. It takes 18.33hours 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.

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

![Boxplot of latency](image1)
![Boxplot of battery lifetime](image2)
![Boxplot of reliability](image3)

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 ($x > q_2 + 1.5 \times (q_2 - q_1)$ or $x < q_1 - 1.5 \times (q_2 - q_1)$); whiskers indicate the range that excludes outliers.

Our method can consistently outperform 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 $L_{dis}$ 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, 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 170ms, 210days, 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 213.76days, and a median reliability of 100%, which meet all given requirements. Similarly, the latency, battery lifetime, and reliability requirements are 165ms, 224days, 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.
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 163.33 ms, 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 130 ms, 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 191.40 ms, a median battery lifetime of 204.74 days, and a median reliability of 94.00%, which fail to meet the latency and reliability requirements.

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.

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.

Figure 9: Accuracy comparison among different methods under different wireless conditions.
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 \((D^P)\) 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 \(L_{\text{cls}}\), the distillation loss \(L_{\text{dis}}\), and the domain-consistent loss \(L_{\text{mmd}}\). 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 \((L_{\text{dis}}: 28.22\%, L_{\text{mmd}}: 26.81\%, \text{and } L_{\text{cls}}: 41.21\%)\). The accuracy is also very low (36.84%) when our method uses \(L_{\text{dis}}\) and \(L_{\text{mmd}}\) due to the critical need of the classification loss on the target domain. The accuracy increases to 64.60% when our method combines \(L_{\text{cls}}\) with \(L_{\text{dis}}\), because the distillation loss \(L_{\text{dis}}\) 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).

### 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 (DGRAM) [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%). DGRAM 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 DGRAM-E by extending DGRAM by allowing an user to specify different PRRs on different channels for each link, and then integrate it with TOSSIM. DISTANCE in NS-3 allows its user to specify the locations of all wireless devices and use the shadowing model to determine packet receptions [62]. 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 [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 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 tied 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 pre-recorded 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, our 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


