Configuring Industrial Wireless Mesh Networks via Multi-Source Domain Adaptation

Xia Cheng
Florida International University
Miami, Florida, USA
xchen075@fiu.edu

Mo Sha
Florida International University
Miami, Florida, USA
msha@fiu.edu

Dong Chen
Colorado School of Mines
Golden, Colorado, USA
dongchen@mines.edu

ABSTRACT

Low-power wireless mesh networks (WMNs) have been widely deployed to connect sensors, actuators, and controllers in industrial facilities. As industrial WMNs become increasingly heterogeneous and complex, recent research has reported that resorting to advanced machine learning techniques to configure WMNs presents significant performance improvements compared to traditional methods. However, it is costly to collect sufficient data to train good network configuration models in many industrial facilities. In such scenarios, the benefits of using learning-based methods that depend on a large amount of data are outweighed by the costs. Recently there have been growing interests in using simulations to configure WMNs because simulations can be set up in less time and introduce less overhead. Unfortunately, recent studies show that the network configuration selected from a simulated network may not be able to help its corresponding physical network achieve desirable performance due to the simulation-to-reality gap. In this paper, we formulate the network configuration prediction as a multi-source domain adaptation problem and introduce a novel solution. Experimental results show that our solution effectively closes the simulation-to-reality gap and provides 80.45% prediction accuracy when it uses cheaply generated simulation data and 440 data traces collected from the physical network for training. As a comparison, the deep neural network (DNN) model trained without using simulation data requires 3,080 costly physical data traces to achieve 80.39% prediction accuracy.

CCS CONCEPTS

• **Networks** → **Wireless local area networks**; **Network management**; **Network simulations**; **Network performance modeling**; **Network measurement**; • **Computing methodologies** → **Machine learning approaches**.

KEYWORDS

Industrial Wireless Mesh Networks, Network Configuration, Multi-Source Domain Adaptation

1 INTRODUCTION

Industrial Internet of Things (IoT) promises one of the largest potential economic effects of IoT – up to $47 trillion in added value globally by 2025, according to the McKinsey report on future disruptive technologies [35]. Industrial wireless mesh networks (WMNs), the underlying support of industrial IoT, typically connect sensors, actuators, and controllers in industrial facilities [31]. Over the last decade, the networks that implement the IEEE 802.15.4-based WMN standards, such as WirelessHART [56], ISA100 [22], WIA-FA [20], and 6TiSCH [21], have been widely deployed in various industrial facilities including manufacturing plants, steel mills, and oil refineries. A decade of real-world deployments has demonstrated the feasibility of using low-power wireless technology to achieve reliable communication in industrial facilities.

Although WMNs achieve good performance most of the time thanks to decades of research, they are difficult to configure, because configuring an industrial WMN is a time-consuming, complex process, which involves theoretical computation, simulation, and field testing, among other tasks. If the network or the application requirement changes, the field engineers may have to repeat the whole network configuration process. As industrial WMNs become increasingly heterogeneous and complex, a breadth of recent research has reported that resorting to advanced machine learning techniques to configure WMNs presents significant performance improvements compared to traditional methods [36, 57, 59]. However, it is very costly to collect sufficient data to train good network configuration models in industrial facilities. In such scenarios, the benefits of using learning-based methods that depend on a large amount of training data are outweighed by the costs. To address the issue, there have been growing interests in using network simulations to configure physical networks [29, 47] because a simulation can be quickly implemented and set up, introduces little to no communication overhead, and allows for different configurations to be evaluated under exactly the same conditions. However, Shi et al. show that the network configuration selected from simulations may fail to help the physical network achieve its desirable performance due to the **simulation-to-reality gap** and propose a single-source domain adaptation method, namely SDA, to narrow the gap [46]. Unfortunately, SDA cannot close the gap when using the data generated by a single simulator and leaves a more than 10% accuracy gap.

In this paper, we present an empirical study to better understand the simulation-to-reality gap in network configuration and introduce *MARIA*, a **M**ulti-source domain **A**daptation solution for wi**R**eless network conf**I**gur**A**tion, which uses a large amount of simulation data together with a small amount of physical data to close the gap. To our knowledge, this paper represents the first study that explores the benefit of using the data generated by multiple simulators to configure industrial WMNs. Specifically, we make the following contributions:

- We present an empirical study that investigates the benefit of using the data produced by multiple simulators to train network configuration models;
- We formulate the network configuration prediction as a multi-source domain adaptation problem and develop MARIA to close the simulation-to-reality gap in network configuration;

- We develop a new method that selects simulation data sets for MARIA to deliver best performance;
- We implement MARIA and evaluate it using four simulators and a physical network that consists of 50 devices. Experimental results show that MARIA provides 80.45% prediction accuracy when using 6,600 simulation data traces together with 440 physical data traces for training. As a comparison, the deep neural network (DNN) model trained without using simulation data must use a large amount of physical data (3,080 data traces) to achieve similar prediction accuracy (80.39%).

The remainder of our paper is organized as the following sections. Section 2 introduces the background of WirelessHART networks and the data sets used in our empirical study and evaluation. Section 3 presents our empirical study. Section 4 and Section 5 introduce the design of MARIA and our simulation data selection method. Section 6 evaluates MARIA. Section 7 reviews the related work. Section 9 concludes the paper.

2 BACKGROUND AND DATA SETS

In this section, we first introduce the background of WirelessHART networks and then present the data sets, which are used in our empirical study and evaluation.

2.1 WirelessHART Networks

In this paper, we use the configuration of WirelessHART networks [56] as an example to present our empirical study and network configuration solution. Today the networks that implement the WirelessHART standard are the most widely used in industrial facilities. For instance, Emerson Process Management, one of the leading WirelessHART network suppliers, has deployed more than 54,835 WirelessHART networks globally and gathered 19.7 billion operating hours of experience [15]. Typically, a WirelessHART network consists of a gateway, multiple access points, and a set of field devices. The network manager, a software module that runs on the gateway, is responsible for performing the management operations, such as collecting link statistics, generating routes, and scheduling transmissions. To meet the stringent real-time and reliability requirements posed by industrial IoT applications, WirelessHART adopts the IEEE 802.15.4 physical layer and employs the time slotted channel hopping (TSCH) technique in the medium access control (MAC) layer. TSCH is designed to combat narrow-band interference and multi-path fading by combining time-slotted medium access, multi-channel communication, and channel hopping. Under TSCH, time is divided into slices of fixed length (e.g., 10ms) that are grouped into a slotframe. Each time slot is long enough to transmit a data packet and an acknowledgement between a pair of communicating devices. All network devices are time synchronized and share the notion of a slotframe that repeats over time. A WirelessHART network uses up to 16 channels and all devices perform channel hopping in each time slot. WirelessHART supports both source routing and graph routing. For each data flow, source routing provides a single route between source and destination, while graph routing provides a primary route and a set of backup routes to improve the network reliability by taking advantage of route diversity. Therefore, each network device is required to have at least two outgoing routes under graph routing.

2.2 Configuration-Performance Data Sets

In this paper, we use the data shared by Shi et al. [46]. The data consists of five data sets: $\mathcal{D}_p$, $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$. $\mathcal{D}_p$ contains the data traces collected from a physical network with 50 TelosB motes [44], which runs the open-source WirelessHART implementation [55] and has six data flows with different sources, destinations, data periods, and priorities. Three performance metrics, including the end-to-end latency L, the battery lifetime B, and the end-to-end reliability E, are selected as the requirements for configuring the WirelessHART network. To meet such performance requirements, three configurable network parameters, including the packet reception ratio (PRR) threshold for link selection R, the number of physical channels used in the network C, and the number of maximum transmission attempts per packet A, are used to generate the routes and transmission schedule for the network operation. When $R \in \{0.60, 0.61, ..., 0.90\}$, $C \in \{1, 2, ..., 8\}$, and $A \in \{1, 2, 3\}$, there exist 744 ($31 * 8 * 3$) parameter combinations. The network manager may generate the same routes and transmission schedule for different network parameter combinations. After removing the redundant configurations that result in the same routes and transmission schedule, there are 88 distinct network configurations. The experiments are performed under each of 88 network configurations and the network performance (L, B, and E values) is measured and stored as a data trace every 50s. Under each network configuration, 75 network performance traces are collected, resulting in 6,600 ($75 * 88$) data traces in total. The same WirelessHART implementation and settings are adopted to perform simulations in the TOSSIM simulator [50], the ns-3 simulator [37], the Cooja simulator [9], and the OMNeT++ simulator [38] to create $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$, respectively. $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$ contain the simulated network performance traces under each network configuration gathered from these four simulators. Each of $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$ also contains 6,600 data traces (75 traces under each of 88 configurations).

3 EMPIRICAL STUDY

In this section, we first formulate the configuration of an industrial WMN as a machine learning problem and introduce our experimental setup. We then present our empirical study that explores the benefit of using multiple simulators to close the simulation-to-reality gap in network configuration.

3.1 Problem Formulation

The primary purpose of configuring an industrial WMN is to select the network configuration, which can help the network achieve its desirable performance. Therefore, the network configuration problem can be formulated as a machine learning problem with the goal of learning a nonlinear mapping $f_\theta(\cdot) : \mathbf{x} \rightarrow \mathbf{y}$, where $\mathbf{x}$ is an input vector of application-specified performance requirements and $\mathbf{y}$ is a vector of network configuration parameters, which allows the network to meet the performance requirements $\mathbf{x}$. Here, we let $\mathbf{x}$ = $concatenation(L, B, E)$ and $\mathbf{y}$ = $concatenation(R, C, A)$. θ denotes the model parameters that are learned from the training data in a

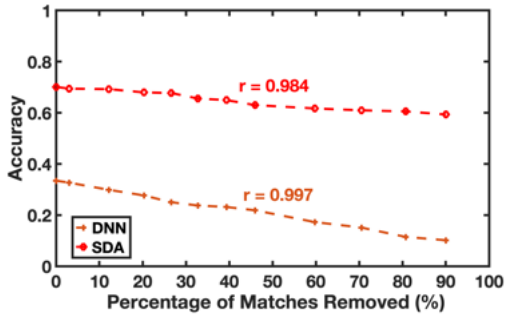


Figure 1: Prediction accuracy when different numbers of matches are removed.

data-driven manner. The network configuration parameter values $\mathbf{y}$ can be discretized without losing the generality. Therefore, f_θ can be further restricted as a discriminative model, which solves a classification problem: the classifier f_θ predicts the network configuration $\mathbf{y}$, which allows the network to meet the performance requirements $\mathbf{x}$.

3.2 Experimental Setup

The primary goal of our empirical study is to answer the question: *Whether using the data produced by multiple simulators can better close the simulation-to-reality gap in network configuration than relying on a single simulator.*

We use $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_p$ (see Section 2.2) in our study. To facilitate the comparisons among $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_p$, we preprocess the data by discretizing the performance measurements $\mathbf{x}$. Specifically, we divide each performance metric into a set of regions and map each performance measurement into one of those regions. For example, all the measurements on the end-to-end latency ranges between 100.12ms and 499.93ms. We divide the latency range $[100, 500)ms$ into 80 regions, map the measurements (e.g., 103.17ms, 290.38ms, and 498.85ms) into those regions, and convert the measurements into the region IDs (e.g., 1, 39, and 80). Each of $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_p$ contains 6,600 data tuples, each of which consists of $\mathbf{x}$ and $\mathbf{y}$. For $\forall(x, y) \in \mathcal{D}_p$, we define the tuple (x, y) as a match in $\mathcal{D}_1$ if (x, y) exists in $\mathcal{D}_1$ and count the number of matches in $\mathcal{D}_1$. We follow the same method to count the number of matches in $\mathcal{D}_2$.

We use the number of matches in each simulation data set as a metric to quantify how much network configuration knowledge learned from it can be applied in $\mathcal{D}_p$, because there exists a strong positive correlation between the number of matches in a data set and the prediction accuracy provided by the machine learning models trained with it. We have performed experiments to confirm this. Specifically, we remove different numbers of matches in a simulation data set, train network configuration models using it, and then measure the prediction performance. For example, Figure 1 shows the prediction accuracy provided by DNN and SDA models on $\mathcal{D}_p$ when the percentage of matches removed from $\mathcal{D}_1$ varies from 0% to 90%. Both DNN and SDA provide lower prediction accuracy when more matches are removed from $\mathcal{D}_1$. The prediction accuracy achieved by DNN is 33.45% when no match is removed from $\mathcal{D}_1$. The prediction accuracy decreases to 10.15% when 90% of matches are removed. Similarly, the prediction accuracy provided by SDA decreases from 70.02% to 59.31% when the percentage of matches removed from $\mathcal{D}_1$ increases from 0% to 90%. More importantly, we observe very high correlation coefficients between the number of matches and the prediction accuracy. The Pearson correlation coefficient under DNN is 0.997 and the one under SDA is 0.984. The results show that the number of matches in a simulation data set is a good metric to reflect how much network configuration knowledge learned from it can be applied in the physical data.

3.3 Results and Observations

To better understand the simulation-to-reality gap in network configuration, we count the number of matches under each network configuration and divide each number by the number of data tuples under a configuration (75) to calculate the percentages of matches. Figure 2 plots the percentage of matches under each network configuration (from 1 to 88). As Figure 2 shows, $\mathcal{D}_1$ does not have any matches under 29 network configurations. For example, $\mathcal{D}_1$ does not have any matches under Configuration 70, thus it is very unlikely for the machine learning model trained with $\mathcal{D}_1$ to make good predictions under this configuration. $\mathcal{D}_2$ also does not contain any matches under 29 configurations. The small percentages of matches under more than half of 88 network configurations explain the cause of the simulation-to-reality gap in network configuration. The network configuration model learned from simulations cannot work well in a physical network because of the domain discrepancy. Such domain discrepancy results from the fact that the theoretical models adopted by the simulators cannot precisely capture extensive uncertainties and variations such as interference in real-world deployments. To investigate whether it is beneficial to use the data produced by multiple simulators for training, we combine $\mathcal{D}_1$ and $\mathcal{D}_2$, count the matches in the combined set $\mathcal{D}_{1+2}$, and compare them against the ones in $\mathcal{D}_1$. We divide the increased matches under each network configuration by 75 to compute the increased percentage of matches. Figure 3 plots the increased percentage of matches under each configuration. As Figure 3 shows, compared to $\mathcal{D}_1$, $\mathcal{D}_{1+2}$ contains more matches under 10 network configurations. For example, $\mathcal{D}_{1+2}$ gets five more matches (6.67%) under Configuration 2. More importantly, under Configuration 79 and 85, $\mathcal{D}_{1+2}$ contains matches (79: 4.0%; 85: 5.33%) while $\mathcal{D}_1$ does not have any matches. $\mathcal{D}_2$ is generated by ns-3, which adopts a theoretical model different from that of TOSSIM. Such a model provides the knowledge, which cannot be learned from $\mathcal{D}_1$. Therefore, $\mathcal{D}_1$ and $\mathcal{D}_2$ provide complementary knowledge on network configuration.

Observation 1*: The data produced by multiple simulators carries more matches than a single simulation data set, which can help on better closing the simulation-to-reality gap in network configuration.*

As Figure 3 shows, the combined set does not have any matches under some network configurations. This emphasizes the importance of using the data collected from the physical network to learn the missing knowledge. Therefore, it is important to develop a solution that makes good use of the data generated by multiple simulators and the one collected from the physical network. A naive solution is to combine the data generated from multiple simulators and use it as the single source domain for domain adaptation. To investigate the performance of such an approach, we use half data from the combined set as the training data and employ SDA to

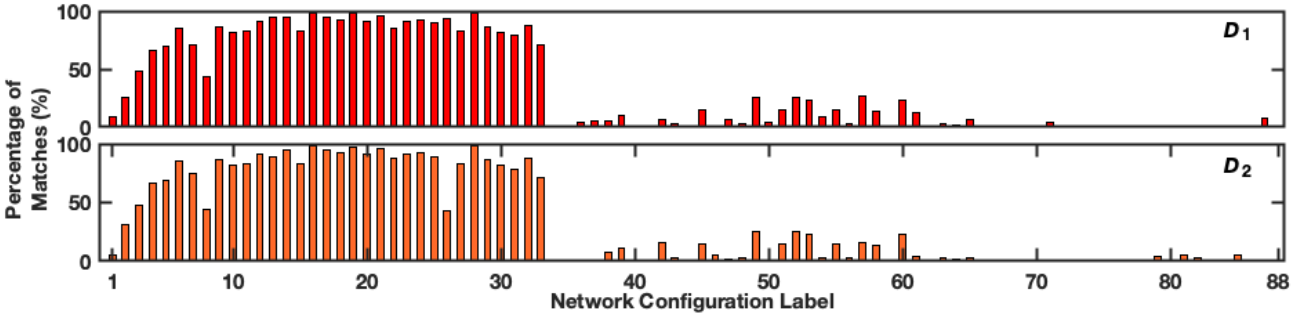


Figure 2: The percentage of matches under each network configuration in $\mathcal{D}_1$ or $\mathcal{D}_2$.

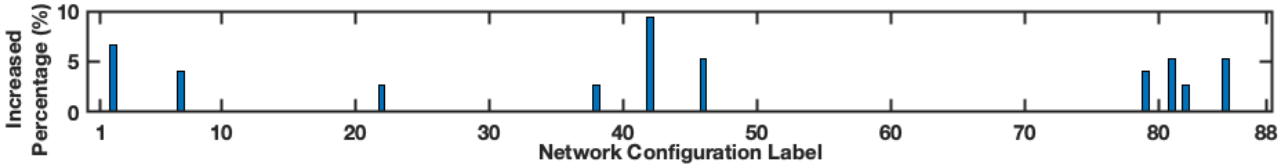


Figure 3: The increased percentage of matches under each network configuration in $\mathcal{D}_{1+2}$.

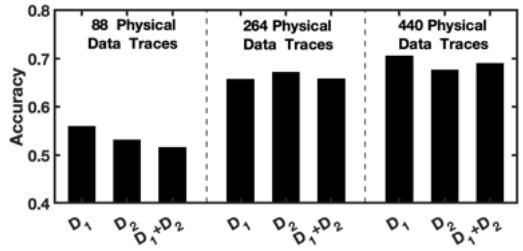


Figure 4: Prediction accuracy on physical data when using different simulation data sets.

perform training. Figure 4 plots the prediction accuracy when we use the combined $\mathcal{D}_1$ and $\mathcal{D}_2$, together with different amounts of data traces collected from the physical network for training. As Figure 4 shows, SDA cannot take the advantage of the combined data and its prediction accuracy when using the combined data is lower than the one when only using $\mathcal{D}_1$ or $\mathcal{D}_2$. For example, when SDA uses $\mathcal{D}_1$, $\mathcal{D}_2$, and 88 data traces (one data trace under each of 88 configurations) collected from the physical network for training, SDA achieves 51.67% accuracy. As a comparison, it provides 55.98% accuracy without using $\mathcal{D}_2$ and 53.22% accuracy by using $\mathcal{D}_2$.

***Observation 2**: Simply mixing the data generated by multiple simulators for domain adaptation cannot effectively improve accuracy.*

4 MARIA

Motivated by our observations in Section 3.3, we develop MARIA, a multi-source domain adaptation solution for wireless network configuration, which uses a large amount of simulation data generated by multiple simulators together with a small amount of physical data to close the simulation-to-reality gap. Specifically, we consider M source domains (the data produced by M simulators): $\mathcal{D}_1^s$, $\mathcal{D}_2^s$, ..., $\mathcal{D}_M^s$, and one target domain $\mathcal{D}^t$ (the data collected from the physical network). We name source domain as simulation domain and target domain as physical domain in the rest of this paper. Each simulation domain consists of data traces $\mathcal{D}_k^s = \{(x_i^s, y_i^s)\}_{i=1}^{N_k^s}$, $k = 1, 2, ..., M$, where x_i^s is the i-th input vector of performance requirements, y_i^s is the corresponding network configuration label, and N_k^s is the number of the data traces in the k-th simulation domain. The physical domain consists of data traces $\mathcal{D}^t = \{(x_i^t, y_i^t)\}_{i=1}^{N^t}$, where N^t is the number of the data traces in the physical domain. The creation of $\mathcal{D}^t$ is much more costly than creating $\mathcal{D}_k^s$, therefore our goal is to learn a good classifier f_θ from the data traces in multiple simulation domains and a little physical domain data ($N^t \ll N_k^s$).

Inspired by the efforts that employ multi-source domain adaptation in surface electromyography physiological signal processing [7, 48], video concept detection [14], image classification [51] as well as the theoretical analysis [4, 34], MARIA trains the classifier f_θ based on the weighted simulation domain data and a few physical domain data traces. Specifically, the classifier f_θ is learned by optimizing the following loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_{physical} + \alpha \mathcal{L}_{simulation} \tag{1}$$

where θ denotes the parameters of the classifier learned during the minimizing loss process, α is a weighting factor, which is used to achieve the balance between the loss on physical domain data $\mathcal{L}_{physical}$ and the loss on simulation domain data $\mathcal{L}_{simulation}$.

Classification loss on physical domain $\mathcal{L}_{physical}$: $\mathcal{L}_{physical}$ allows the classifier to learn from a small amount of physical domain data by employing the cross-entropy loss:

$$\mathcal{L}_{physical} = - \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}^t} y \log(f_\theta(x)) \tag{2}$$

where $\mathbf{y}$ is the one-hot label and $f_\theta(x)$ is the prediction provided by the classifier.

Domain Alignment loss $\mathcal{L}_{simulation}$: $\mathcal{L}_{simulation}$ helps the classifier learn from a larger amount of the simulation data generated by different simulators. Although the distribution of data traces in each simulation domain is different from the one in the physical domain, each simulation domain shares a few matches with the

physical domain. Therefore, the classifier can learn more knowledge on the input feature space and the corresponding classification label of the physical domain when training with the matches of different simulation domains.

Motivated by the observation that different simulation domains share different numbers of matches with the physical domain (as Figure 2 shows), MARIA employs a set of weighting factors to differentiate the contributions of different simulation domains to the loss $\mathcal{L}_{simulation}$ and uses such a weighting scheme to ensure the better use of those simulation domains that contain more matches and less use of the simulation domains that contain many duplicated even conflicted data traces in the training process. Specifically, MARIA uses the maximum mean discrepancy (MMD) criterion proposed in [5] to measure the relevance between each simulation domain and the physical domain. The core idea of MMD is to match two distributions based on the mean of features in the reproducing kernel Hilbert space (RKHS) after mapping them to RKHS. By computing MMD between each simulation domain and the physical domain, MARIA defines the weighting factor β_i and assigns it to its corresponding cross-entropy loss. Finally, $\mathcal{L}_{simulation}$ is decided by calculating the function:

$$\mathcal{L}_{simulation} = -\sum_{i=1}^{M} \beta_i \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in\mathcal{D}_i^s} y \log(f_\theta(x)) \tag{3}$$

where β_i denotes the relevance between $\mathcal{D}_i^s$ and $\mathcal{D}^t$. To compute β_i, MARIA first measures the MMD value between each simulation domain and the physical domain, applies the exponential function to each measured value, and then adds them up. Finally, β_i is calculated according to the ratio of the single MMD value to the sum of all values. Specifically, β_i is adjusted by calculating the following function:

$$\beta_i = \frac{\exp(-\gamma(Dis(\mathcal{D}_i^s, \mathcal{D}^t))^\delta)}{\sum_{i=1}^{M} \exp(-\gamma(Dis(\mathcal{D}_i^s, \mathcal{D}^t))^\delta)} \tag{4}$$

where $Dis(\mathcal{D}_i^s, \mathcal{D}^t)$ denotes the measured MMD value between $\mathcal{D}_i^s$ and $\mathcal{D}^t$, γ and δ are the coefficients to adjust the spread of $Dis(\mathcal{D}_i^s, \mathcal{D}^t)$.

Our implementation adopts a DNN architecture that consists of three fully connected layers. It uses L, B, and E as the input features of x_i, two hidden layers, and the rectified linear unit (ReLU) as the activation function for those hidden layers. In addition, our implementation sets the number of neurons in the output layer to 88 (equal to the number of all distinct network configuration categories) and employs the softmax function as the activation function for the output layer. While considering the number of total data traces, our implementation sets the batch size equal to the number of data traces and updates the parameters θ once in each epoch to speed up the training process. Our implementation employs the Adam optimizer [24] to optimize the parameters θ and configures the learning rate to 0.12. We set $\gamma = 1000$ and $\delta = 2$ when we compute β_i for each simulation domain before optimizing the loss function $\mathcal{L}(\theta)$.

5 SIMULATION DATA SELECTION

MARIA is designed to train network configuration models based on input simulation and physical data. However, it is not always beneficial to use all available data for training because some simulation data sets may not carry unique network configuration knowledge. Moreover, having more simulation domains increases the difficulty of optimizing a good network configuration model.

Inspired by the insights learned in Section 3.3, we develop a method that selects the simulation data sets for training based on a small amount of physical domain data in $\mathcal{D}^t$. To reduce the difficulty of optimizing the loss, our method minimizes the number of simulation domains used for training and makes sure that the selected simulation data sets include all matches. Specifically, we consider M sets: T_k, $k = 1, 2, ..., M$, where T_k is associated with the k-th domain $\mathcal{D}_k^s$ and includes all matches when comparing $\mathcal{D}_k^s$ with $\mathcal{D}^t$. The number of sets M is equal to the number of simulation domains. The union of those M sets contains all matches in all simulation data, namely the universe. Therefore, the simulation domain selection problem can be formulated as the set cover problem, which aims to identify the smallest sub-collection of those M sets whose union equals the universe. We have proved the problem to be NP-hard and omitted the proof here due to space limit.

Algorithm 1: Simulation Data Selection Method

Input : $T_1, T_2, ..., T_M$
Output: a sub-collection

Initialize $n = 0$, $count[] = \{0\}$, $S = \{T_1, T_2, ..., T_M\}$;
$\forall p, q \in [1 .. M]$, initialize $U_p^q = T_p \cap T_q$ if $p \neq q$; otherwise $U_p^q = \varnothing$;
for $i \leftarrow 1$ *to* M **do**
  for $j \leftarrow 1$ *to* M **do**
    If $U_i^j \neq \varnothing$, $count[i] = count[i] + 1$;
  If $T_i = \bigcup_{j=1}^{M} U_i^j$, $n = n + 1$;
for $n \geq 1$ **do**
  if $n > 1$ **then**
    if $\exists T_p \subseteq T_q$ *and* $T_p, T_q \in S(p, q \in [1 .. M], p \neq q)$ **then**
      $S = S - \{T_p\}$;
    else
      Identify T_p among the n sets, where $count[p]$ is the least ;
      $S = S - \{T_p\}$;
  else
    $S = S - \{T_p\}$; // T_p is the only one
  $\forall r \in [1 .. M]$, $U_p^r = U_r^p = \varnothing$;
  $n = 0$, $count[] = \{0\}$;
  for $i \leftarrow 1$ *to* M **do**
    for $j \leftarrow 1$ *to* M **do**
      If $U_i^j \neq \varnothing$, $count[i] = count[i] + 1$;
    If $T_i \in S$ and $T_i = \bigcup_{j=1}^{M} U_i^j$, $n = n + 1$;
Output S;

Our simulation data selection method removes each simulation data set, which does not contain unique matches, from the initial

M-set collection one by one. Algorithm 1 illustrates our method with a collection of simulation data sets T_1, T_2, ..., T_M as its input. Algorithm 1 first initializes a counter n for the number of redundant sets, an array *count*, and a collection of sets S (line 1). It also initializes the intersection between each pair of sets U (line 2). Then, it traverses each pair of sets in a two-level nested loop and counts the number of non-empty intersections for each set (line 3-5). If a set does not contain unique matches, this set is a candidate that can be removed from S and n increases by one (line 6). There may be more than one candidate. If any set T_p is a subset of T_q, T_p is removed from S to eliminate the inclusion relation (line 9-10). When a redundant set is removed, the sets that represent this redundant set may have to be kept in the final sub-collection. To minimize the final sub-collection, Algorithm 1 sorts *count* and removes T_p with the least value in *count* among n candidates (line 12-13). If T_p is the only candidate, it is removed (line 15). Then, the intersections associated with T_p are emptied (line 16). Accordingly, the counter n and *count* are updated to check whether there still exists any redundant set (line 18-21). Finally, the collection S is the output (line 22). Therefore, the simulation data sets associated with the sets in S are selected for MARIA.

6 EVALUATION

We perform a series of experiments to examine the performance of MARIA on identifying good network configurations. We first evaluate the prediction accuracy of MARIA and compare it against the one provided by the state-of-the-art method SDA [46] (see Section 6.1). We change the data used by SDA to create multiple baselines. We then apply the network configurations selected by MARIA on a physical network and measure its performance (see Section 6.2). In addition, we investigate the effects of different loss functions and simulation domain data size on the performance of MARIA (see Section 6.3 and 6.4) and study how our simulation data selection affects MARIA (see Section 6.5). Finally, we validate the performance of MARIA by introducing a validation stage (see Section 6.6).

We run MARIA and baselines on the data introduced in Section 2.2. Specifically, we use $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$, which are generated by TOSSIM, ns-3, Cooja, and OMNeT++, respectively, for training. We divide 6,600 physical domain data traces into two sets: 3,960 traces (60% of the data) for training and 2,640 traces (40% of the data) for testing. We train the network configuration model using the simulation domain data and the training set of the physical domain data, and evaluate the model using the testing set of the physical domain data. We implement our neural network under the framework provided by PyTorch. In each iteration, our neural network is trained with all data in the training set and all parameters are updated accordingly. We employ the program provided by Shi et al. [46] as our baseline implementation and adopt its default settings.

6.1 Performance of MARIA

We first evaluate the prediction accuracy of MARIA and compare it against the baselines. Our simulation data selection method selects $\mathcal{D}_1$ and $\mathcal{D}_2$ for MARIA as its input. To ensure fair comparisons, we also use $\mathcal{D}_1$ and $\mathcal{D}_2$ for SDA and vary the data used as the single simulation domain to create three baselines: (i) using $\mathcal{D}_1$ (named SDA-TOSSIM), (ii) using $\mathcal{D}_2$ (named SDA-ns-3), and (iii) using the combination of $\mathcal{D}_1$ and $\mathcal{D}_2$: half from $\mathcal{D}_1$ and half from $\mathcal{D}_2$ (named SDA-mixed). We define 88 data traces (one data trace under each network configuration) as one shot of data. Figure 5 plots the prediction accuracy of different methods when one to 10 shots of physical domain data, together with simulation domain data, are used for training. When only one shot of physical domain data (88 data traces) is used for training, the prediction accuracy provided by MARIA is 58.18%. Our three baselines provide similar performance. The highest accuracy provided by three baselines is 55.98%. None of these methods can perform well when they use a single shot of physical domain data for training. This confirms the observation that there exist significant performance variations when the network uses the same configuration due to runtime dynamics and multiple shots of physical data are needed for effective domain adaptation [45]. The results also show that blindly exploring the parameter space (e.g., using a brute-force method) would require a large amount of physical domain data (thousands of experimental runs on the physical network) to identify a good configuration. Collecting one sample under each configuration is not enough to produce a good model. After more physical domain data is used for training, the prediction accuracy of all methods increases. As Figure 5 shows, MARIA consistently achieves the best performance. For example, when using three shots of physical domain data, MARIA provides 75.68% prediction accuracy, while SDA-TOSSIM, SDA-ns-3, and SDA-mixed provide 65.76%, 67.15%, and 64.85% accuracy, respectively. When five shots of physical domain data (440 data traces) are used for training, MARIA achieves 80.45% prediction accuracy, which is close to the 80.39% accuracy provided by the DNN model that is trained with 35 shots of physical domain data (3,080 data traces). As a comparison, SDA-TOSSIM, SDA-ns-3, and SDA-mixed provide 70.56%, 67.72%, and 69.75% accuracy, respectively, when five shots of physical domain data are used for training. The reason behind is that different simulators employ different models to capture the effects of ambient environments and the network configuration knowledge offered by different simulators is complementary to each other. MARIA outperforms the baselines thanks to its capability of taking advantage of the diverse network configuration knowledge offered by multiple simulators.

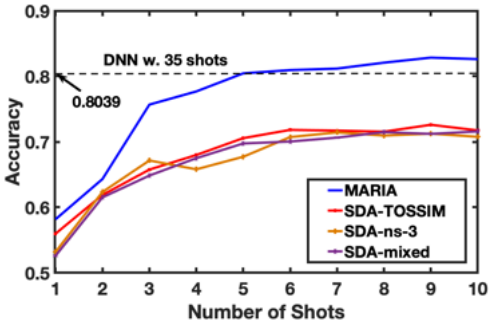


Figure 5: Prediction accuracy of different methods when one to 10 shots of physical domain data, together with the simulation domain data, are used for training. The black dotted line indicates the accuracy provided by the DNN model trained with 35 shots of physical domain data.

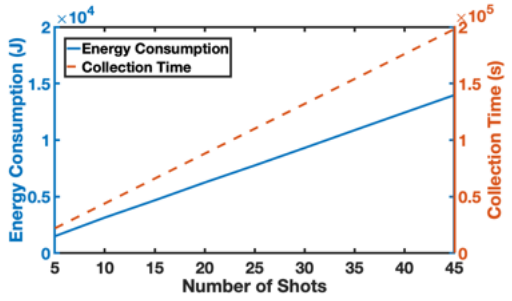


Figure 6: Energy consumption and time consumption of collecting different amounts of physical data from a physical network with 50 devices.

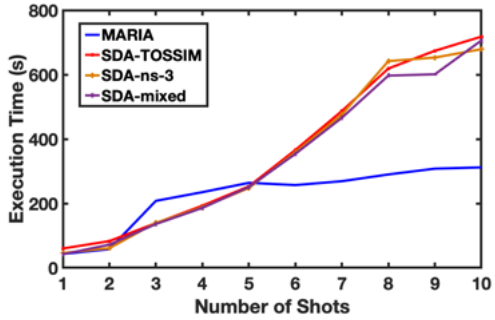


Figure 7: Time consumption of different methods when different amounts of physical data are used for training.

Using a small amount of physical domain data to provide a good model represents a very important feature of MARIA because the data collection from a physical network is very time and energy consuming. Figure 6 plots the energy and time consumption of collecting physical domain data from a network with 50 devices. As Figure 6 shows, collecting five shots of physical domain data consumes 1,503J and takes 6.11$hours$, while the collection of 10 shots of physical domain data consumes 3,150J and takes 12.22$hours$. As a comparison, it consumes 13,974J and takes 54.99$hours$ to collect 45 shots of physical domain data to train the DNN model for the 82.92% accuracy. As Figure 5 shows, the prediction accuracy of all methods increases slowly when more than five shots of physical domain data are used for training. For example, the accuracy provided by MARIA increases slightly from 80.45% to 82.70% and the accuracy provided by SDA-TOSSIM, the one providing the best performance among three baselines, increases from 70.56% to 71.74%, when we increase the physical domain data used for training from five shots to 10 shots. Besides, the low accuracy provided by SDA-mixed also shows that simply mixing the data generated by different simulators for training does not work well. This is because many duplicated even conflicted data traces are also added into the training data, which significantly increases the difficulty of optimizing the classifier.

MARIA is designed to be lightweight. We then evaluate the efficiency of MARIA by measuring its runtime overhead. We run all methods on a server equipped with a 2.6GHz quad-core processor and measure the training time. Figure 7 plots the time consumption of different methods when different amounts of physical domain data are used for training. As Figure 7 shows, MARIA consumes slightly more time than the baselines when three, four, and five shots of physical domain data are used for training. The slight increases in time consumption are in exchange for a proportionally much-larger increase in prediction accuracy. More importantly, the time consumption of MARIA increases slowly when more physical domain data is used for training, while the time consumption of all baselines increases sharply. For example, the time consumed by MARIA increases from 264s to 312s when we increase the physical domain data from five shots to 10 shots. As a comparison, the time consumed by SDA-TOSSIM increases from 253s to 718s. This represents another very important feature of MARIA, which results from the fact that our model is trained with all data traces of the training set in each iteration while the baselines only use part of the data traces in the training set during each iteration.

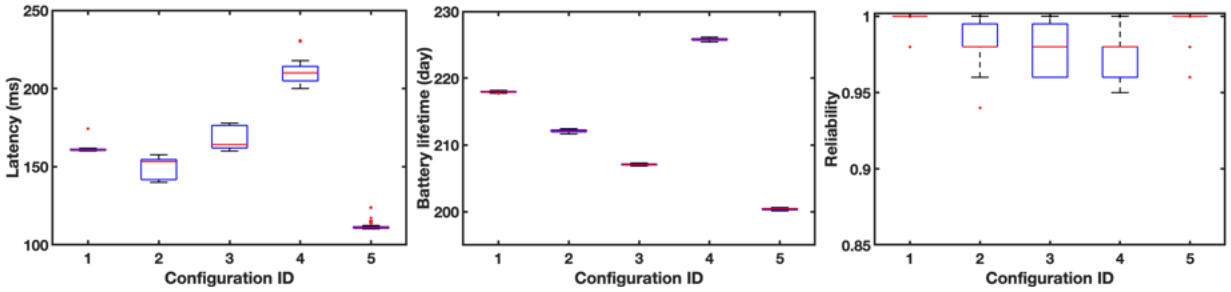
6.2 Validation on a Physical Network

To demonstrate the practicality of MARIA, we apply the network configurations selected by MARIA on a physical network that consists of 50 devices and measure the network performance including the end-to-end reliability, the end-to-end latency, and the battery lifetime. We run the open-source implementation of WirelessHART networks provided by Li et al. [55] and configure multiple data flows. We run the experiments under different network topologies by varying the number of data flows and the locations of sensors and actuators. When performing the experiments, we first inject different network performance requirements into MARIA, apply the network configurations selected by our model on the physical network, and then examine whether the network that uses the selected parameters can provide the desirable performance. Table 1 lists the network configurations selected by our model based on five different sets of network performance requirements. We repeat experiments under each network configuration 100 times on the physical network. Figure 8 plots the boxplots of latency, battery lifetime[1], and reliability when deploying five sets of network configurations selected by our model. As Figure 8 shows, when the network uses the configurations selected by our model, its performance meets the requirements specified by the application (listed in Table 1). For example, the latency, battery lifetime, and reliability requirements are 165ms, 215day, and 98%, respectively, in the first testing case (ID = 1). When employing the network configuration selected by our model (87% as the PRR threshold, three physical channels, and three transmission attempts per packet), the median values of latency, battery lifetime, and reliability measured from the network are 160.80ms, 217.93day, and 100%, respectively, which meet all specified requirements. Similarly, the latency, battery lifetime, and reliability requirements are 180ms, 205day, and 96%, respectively, in the third case (ID = 3). When the network uses the selected configuration (87% as the PRR threshold, four physical channels, and two transmission attempts per packet), the network achieves the median latency of 164.13ms, the median battery lifetime of 207.05day, and the median reliability of 98.0%, which meet all specified requirements. These results show that the network configurations selected by MARIA can help the network achieve desirable performance.

[1]The battery lifetime is calculate according to the assumption that each device is powered by two Lithium Iron AA batteries with a total capacity of 42,700J, the time duration of radio activities, and the power consumption of the radio in each state provided by the radio chip data sheet [10].

Table 1: Five Example Network Configurations Selected by Our Model.

Case ID	Performance Requirement			Network Configuration		
	Latency (ms)	Battery lifetime (day)	Reliability (%)	PRR threshold (%)	# of channels	# of Tx attempts
1	165	215	98	87	3	3
2	160	210	96	90	3	2
3	180	205	96	87	4	2
4	220	225	95	90	7	2
5	120	200	98	84	2	3



(a) End-to-end latency. (b) Battery lifetime. (c) End-to-end reliability.

Figure 8: Measured network performance when the network configurations selected by our model are used. Central mark in box indicates median; bottom and top of box represent the 25th percentile and 75th percentile; red dots indicate outliers; whiskers indicate the range that excludes outliers.

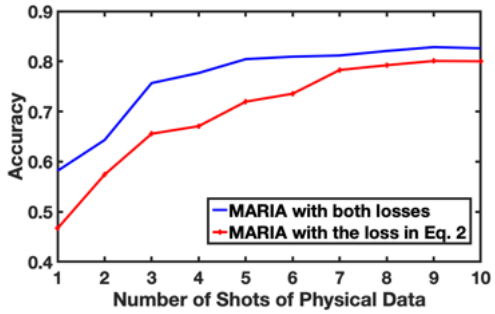


Figure 9: Prediction accuracy provided by MARIA without the loss function $\mathcal{L}_{simulation}$.

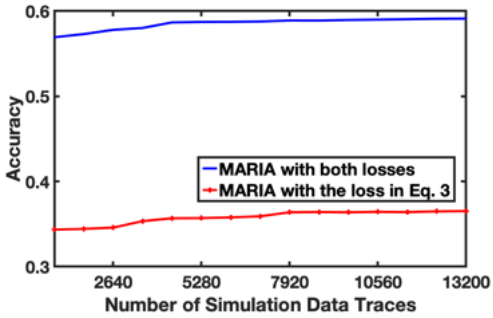


Figure 10: Prediction accuracy provided by MARIA without the loss function $\mathcal{L}_{physical}$.

6.3 Effect of Different Loss Functions

To investigate the effects of different loss functions on the performance of MARIA, we repeat the experiments by disabling one loss function in each run. We first remove all simulation domain data and disable the loss function $\mathcal{L}_{simulation}$ defined in Eq. 3. Figure 9 plots the prediction accuracy when we disable $\mathcal{L}_{simulation}$ and use different shots of physical domain data for training. As Figure 9 shows, without using $\mathcal{L}_{simulation}$, the accuracy provided by MARIA is 71.97% when five shots of physical domain data are used for training. As a comparison, MARIA achieves 80.45% accuracy with both losses enabled. Without using $\mathcal{L}_{simulation}$, MARIA must use five more shots of physical domain data to achieve similar prediction accuracy (80.03%). The results show that the loss function $\mathcal{L}_{simulation}$ plays an important role in enhancing the prediction accuracy, especially when only a small amount of physical domain data is available for training.

We then remove all physical domain data traces and disable the loss function $\mathcal{L}_{physical}$ defined in Eq. 2. Figure 10 plots the prediction accuracy provided by MARIA without using $\mathcal{L}_{physical}$ when different numbers of simulation domain data traces are used for training. As a comparison, we plot the accuracy achieved by MARIA when it uses both loss functions and one shot of physical domain data for training in Figure 10. As Figure 10 shows, the prediction accuracy increases from 34.35% to 36.38% when the number of data traces increases from 880 to 7,920. This is because the number of matches increases when more simulation domain data is used for training. When more than 7,920 simulation data traces are used for training, the accuracy increases slightly as the newly added data includes very few unique matches. For example, the prediction achieved by MARIA without $\mathcal{L}_{physical}$ is 36.52% when 13,200 data traces are used for training. As a comparison, the accuracy provided by MARIA with both loss functions is significantly higher when it uses only one shot of physical domain data. For example, MARIA achieves 56.93% accuracy when it uses 880 simulation data traces for

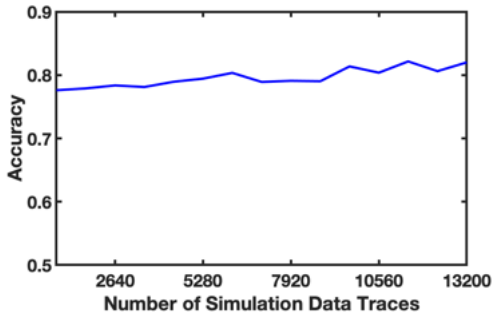


Figure 11: Prediction accuracy when we use different numbers of simulation data traces.

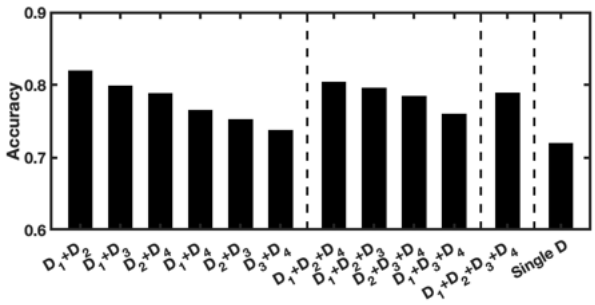


Figure 12: Prediction accuracy when the simulation domains consist of the data sets generated by different simulators.

training and 59.12% accuracy when it uses 13,200 simulation data traces. The results show the significant effect of the loss function $\mathcal{L}_{physical}$ on the performance of MARIA.

6.4 Effect of Simulation Domain Data Size

To study the effect of the size of simulation domain data on the performance of MARIA, we repeat the experiments when different numbers of simulation data traces (a half from $\mathcal{D}_1$ and the rest from $\mathcal{D}_2$) and five shots of physical domain data are used for training. Figure 11 plots the prediction accuracy provided by MARIA when we increase the total number of simulation data traces from 880 to 13,200. As Figure 11 shows, the prediction accuracy increases when we add more simulation data traces into the training set. For example, MARIA achieves 77.6% accuracy when using 880 simulation data traces and 80.34% accuracy when using 6,160 simulation domain data traces. After using more than 6,160 data traces, the prediction accuracy achieved by MARIA increases slowly. For instance, when 13,200 simulation data traces are used for training, the prediction accuracy is 82.00%. This is because the newly added data traces introduce a small amount of unique matches when the size of simulation domain data is large. The results show that MARIA can achieve a high accuracy when sufficient simulation domain data and a few physical domain data traces are used together for training.

6.5 Effect of Our Simulation Data Selection Method

To examine the effect of our simulation data selection method on the performance of MARIA, we disable it and manually create 11 different simulation domain combinations by including some or all of those four data sets (six combinations by including two data sets, four combinations by including three data sets, and one combination by including all four data sets). We then use each of 11 simulation domain combinations together with five shots of physical domain data (440 traces) to train the network configuration model through MARIA. To ensure fair comparisons, we let each simulation domain combination include 13,200 traces. Figure 12 plots the accuracy provided by MARIA when it uses each of 11 combinations and one single simulation domain for training.

As Figure 12 shows, MARIA achieves the best performance (82.00% accuracy) when it uses $\mathcal{D}_1$ (generated by TOSSIM) and $\mathcal{D}_2$ (generated by ns-3) for training. As a comparison, the model that uses $\mathcal{D}_3$ (generated by Cooja) and $\mathcal{D}_4$ (generated by OMNeT++) for training provides 76.55% accuracy. This is because the combination of $\mathcal{D}_1$ and $\mathcal{D}_2$ includes more matches than the combination of $\mathcal{D}_3$ and $\mathcal{D}_4$. The results confirm the correctness of the selection made by our method. Figure 12 also shows that using the data from three or all four simulators does not provide better performance. For instance, MARIA provides 80.38% accuracy when it uses $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_4$ for training and 79.58% accuracy when it replaces $\mathcal{D}_4$ with $\mathcal{D}_3$. The accuracy of our model is only 78.90% when using the data generated by all simulators. The results show that blindly introducing more simulation domains cannot improve the performance of MARIA. This is because the combination of $\mathcal{D}_1$ and $\mathcal{D}_2$ already includes all matches in the simulation data and the difficulty of optimizing the classifier among multiple simulation domains and the physical domain increases when more simulation domains are introduced. The results emphasize the importance of our simulation data selection method. Besides, MARIA provides the lowest prediction accuracy (72.01%) when it uses the data produced by a single simulator for training. The results highlight the benefit of using the simulation data gathered from multiple simulators.

To further validate our simulation data selection method, we vary the data in the simulation data sets, run Algorithm 1 to select simulation data, and compare the selections against the optimal ones. Specifically, we remove different amounts of data from one data set and combine it with the other sets to create various simulation data combinations. Under each combination, we randomly remove a certain ratio of data from one data set 1,000 times and compare the simulation data sets selected by Algorithm 1 against the optimal selections. If the selection provided by our simulation data selection method is the same with the optimal selection, we define it as a correct selection. Figure 13 plots the selection accuracy (the number of correct selections divided by 1,000) when different ratios of data (0% to 90%) are removed from $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$, respectively. As Figure 13 shows, the accuracy is always 100% when the data removal from $\mathcal{D}_1$ ranges from 0% to 70%. After that, the accuracy decreases slightly to 99.90% when 80% of data is removed from $\mathcal{D}_1$ and finally reaches 99.60% when 90% of data is removed from $\mathcal{D}_1$. When more data is removed, the chance for no unique matches in $\mathcal{D}_1$ increases. Similarly, the accuracy decreases from 100% to 99.90% when 60% of data is removed from $\mathcal{D}_2$. It further decreases to 98.10% when 90% of data is removed $\mathcal{D}_2$. As a comparison, the accuracy is consistently 100% when we remove data from $\mathcal{D}_3$ or $\mathcal{D}_4$. They both are proper subsets of $\mathcal{D}_1$ or $\mathcal{D}_2$, therefore the data removal does not result in any change on unique matches. The

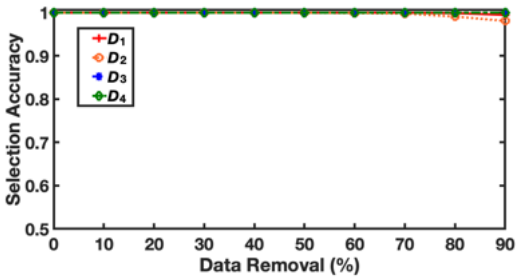


Figure 13: Selection accuracy of our data selection method when we vary data in $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$, respectively.

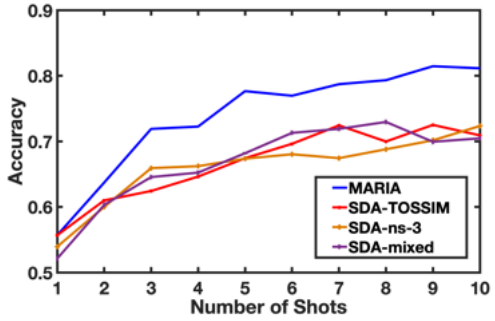


Figure 14: Prediction accuracy with the validation stage.

results demonstrate the effectiveness of our data selection method in selecting the best-suited simulation data sets for MARIA.

6.6 Performance Validation

Finally, we repeat our experiments with a validation stage to further validate the performance of MARIA. We use the same simulation domain data as presented in Section 6.1, and divide 6,600 physical domain data traces into three sets: training set (60% of the data), validation set (20% of the data), and testing set (20% of the data). We train the network configuration model using the simulation data and the training set of the physical domain data, validate the model on the validation set of the physical domain data, and then evaluate the optimal model validated using the testing set of the physical domain data. We randomly select the data samples from the physical domain data and put them into those three data sets and repeat the experiments to eliminate the effect of data partitions on the results. Figure 14 plots the prediction accuracy achieved by MARIA and the baselines on the testing set from an experimental run. As Figure 14 shows, MARIA consistently achieves the best performance. For example, MARIA achieves 77.65% prediction accuracy when five shots of physical domain data (440 data traces) are used for training. As comparisons, SDA-TOSSIM, SDA-ns-3, and SDA-mixed provide 67.42%, 67.35%, and 68.18% accuracy, respectively. We observe similar results when we split the physical data differently. The experimental results confirm that the prediction accuracy provided by MARIA is not resulted from overfitting.

7 RELATED WORK

Significant efforts have been made in the literature to model the characteristics of wireless sensor networks (WSNs) and optimize network configurations through mathematical techniques such as convex optimization [32], game theory [1], and meta heuristics [43]. For example, the characteristics of low-power wireless links have been studied empirically with different platforms, under varying experimental conditions, assumptions, and scenarios [3], and network configuration methods have been developed to improve the performance of WSNs by adapting a few parameters in the physical and media access control (MAC) layers [12, 13, 16, 42, 52, 62]. 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 those traditional methods. For instance, deep learning has been used to automatically uncover correlations that would otherwise have been too complex to be extracted by human experts [6, 26, 36, 59] and reinforcement learning has been employed to enable network self-configuration [11, 23, 28, 33, 40, 53, 57]. The key component behind the remarkable success of those data-driven methods is the capability of optimizing a huge number of free parameters 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 network control targets, such as resource allocation, queue management, and congestion control. However, data collection from many wireless deployments, including the ones in industrial facilities, is costly; therefore it is difficult to obtain sufficient information for deep learning to train a good model or reinforcement learning to identify an optimal policy for network configuration. In such scenarios, the benefits of employing learning-based methods that require much data are outweighed by the costs. To address this issue, there have been increasing interests in using network simulations to configure physical networks [29, 47]. For instance, Liu et al. develop a framework, which integrates the process control system model and the network model into a unified discrete-event simulator and leverages it to identify good network configurations [29]. Slabicki et al. introduce an open-source framework for end-to-end LoRa simulations and propose to dynamically optimize link parameters for scalable and efficient network operations [47]. Unfortunately, a recent study shows that the network configuration selected according to simulations may fail to help the physical network achieve the desirable performance due to the simulation-to-reality gap [46]. This paper aims to close such a gap and provide a new solution that learns a good predictive model for network configuration using a large amount of inexpensive simulation data and a small number of costly physical measurements.

Domain adaptation has been used to narrow the gap between different domains in computer vision [51, 54, 58], natural language processing [39], magnetic resonance imaging [18], network performance modeling [27], structural health monitoring [17], and building occupancy estimation [2, 60]. Domain adaptation aims to learn from one or multiple source domains, together with or without a target domain, and then generate a model that performs well on the target domain. Generally, the source domain data and the target domain data share the same space for both input features and labels, but they do not share the same distribution. Over the

past decades, many single-source domain adaptation methods have been proposed to address the domain shift [25, 30]. However, there have been very few studies looking into the use of domain adaptation to address the domain shift issue in network configuration. Recently, Shi et al. develop SDA, which leverages a teacher-student neural network to close the simulation-to-reality gap in network configuration [46]. However, SDA cannot close the gap when using the data produced by a single simulator and leaves a more than 10% accuracy gap. More recently, Cheng et al. propose a meta-learning based solution, which adapts network configuration at runtime. However, it is not applicable for closing the simulation-to-reality gap [8]. To our knowledge, this paper represents the first systematic effort to explore the benefit of using the data generated by multiple simulators to close the simulation-to-reality gap in network configuration.

Multi-source domain adaptation [49] has been employed recently in computer vision [61], natural language processing [19, 39], and physiological signal processing [7, 48]. For instance, Sun et al. develop a multi-source domain adaptation method, which computes the weighting factors for multiple sources according to both marginal and conditional probability differences between the source domains and the target domain [48]. Duan et al. propose to leverage a set of pre-computed classifiers independently learned from multiple source domains to effectively reduce the domain discrepancy [14]. Peng et al. propose to transfer knowledge learned from multiple source domains to an unlabeled target domain by dynamically aligning moments of their feature distributions [41]. MMD is employed by those learning methods to measure the discrepancy between domains and diminish the distribution shift between the source domain and target domain accordingly [14, 48]. Moreover, early theoretical analysis provides strong guarantees for representing the target domain distribution as the weighted combination of source domain distributions [4, 34]. Inspired by the existing analysis and methods, we develop the first solution that leverages the multi-source domain adaptation to close the simulation-to-reality gap in network configuration. Our experimental results show that our solution can close the simulation-to-reality gap and significantly outperform the state-of-the-art method, SDA.

8 DISCUSSIONS ON REAL-WORLD APPLICATIONS AND GENERALIZATION

This paper aims to provide a solution for engineers to well configure an industrial WMN after they deploy it in the field. We recommend the engineers following six steps to configure the network. First, the engineers should measure the ambient operation environments, such as collecting noise traces. Second, the engineers should implement the physical network in multiple simulators and then feed the environmental measurements into those simulators. Third, the engineers should run simulations in each simulator to measure the performance of the simulated network with every possible combination of network parameters. A large amount of simulation data that carries valuable network configuration knowledge can be inexpensively obtained in this step. Fourth, the engineers should collect a few performance measurements from the physical network when it uses each possible combination of network parameters. Collecting the physical data in this step introduces significant overhead (see Figure 6), which emphasizes the importance of minimizing the amount of physical data needed for training in our solution. Fifth, the engineers should train the network configuration model using MARIA. Finally, the engineers can configure the physical network in the field with the configuration selected by MARIA based on the network performance requirements posed by the upper layer industrial applications.

We expect our solution would affect not only industrial WMNs but other complex wireless networks as it provides a replicable template for novel network configuration strategies. Our data-driven design is not tied to any specific network protocol stack, network topology, or performance metric. Moreover, our deep learning based solution is capable of accepting a large number of tunable parameters and automatically uncovering the correlations between those parameters and network performance that would otherwise have been too complex to be extracted by human experts.

9 CONCLUSIONS

In this paper, we present MARIA, a novel multi-source domain adaptation solution for industrial WMN configuration. Experimental results show that MARIA provides 80.45% prediction accuracy when it uses 6,600 cheaply generated simulation data traces and 440 data traces collected from the physical network for training. As a comparison, the DNN model trained only with physical data requires 3,080 costly physical data traces to achieve comparable prediction accuracy (80.39%).

ACKNOWLEDGMENT

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

REFERENCES

[1] Eitan Altman, Thomas Boulogne, Rachid El Azouzi, Tania Jiménez, and Laura Wynter. 2006. A Survey on Networking Games in Telecommunications. *Computers and Operations Research* 33, 2 (2006), 286–311.
[2] Irvan B Arief-Ang, Flora D Salim, and Margaret Hamilton. 2017. DA-HOC: Semi-Supervised Domain Adaptation for Room Occupancy Prediction Using CO_2 Sensor Data. In *ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys)*.
[3] Nouha Baccour, Anis Koubâa, Luca Mottola, Marco Antonio Zúñiga, Habib Youssef, Carlo Alberto Boano, and Mário Alves. 2012. Radio Link Quality Estimation in Wireless Sensor Networks: A Survey. *ACM Transaction on Sensor Network* 8, 4 (2012).
[4] John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman. 2008. Learning Bounds for Domain Adaptation. In *Advances in Neural Information Processing Systems*.
[5] Karsten M. Borgwardt, Arthur Gretton, Malte J. Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J. Smola. 2006. Integrating Structured Biological Data by Kernel Maximum Mean Discrepancy. *Bioinformatics* 22, 14 (2006), e49–e57.
[6] Xianghui Cao, Lu Liu, Yu Cheng, and Xuemin Shen. 2018. Towards Energy-Efficient Wireless Networking in the Big Data Era: A Survey. *IEEE Communications Surveys & Tutorials* 20, 1 (2018), 303–332.
[7] Rita Chattopadhyay, Qian Sun, Wei Fan, Ian Davidson, Sethuraman Panchanathan, and Jieping Ye. 2012. Multisource Domain Adaptation and Its Application to Early Detection of Fatigue. *ACM Trans. Knowl. Discov. Data* 6, 4 (2012), 26 pages.
[8] Xia Cheng and Mo Sha. 2023. Meta-Learning Based Runtime Adaptation for Industrial Wireless Sensor-Actuator Networks. In *IEEE/ACM International Symposium on Quality of Service*.
[9] Cooja. 2021. Source Code of Cooja. https://github.com/contiki-os/contiki/wiki/An-Introduction-to-Cooja
[10] Crossbow. 2010. TelosB. https://dtsheet.com/doc/1368377/telosb-datasheet---willow-technologies
[11] Hiba Dakdouk, Erika Tarazona, Reda Alami, Raphaël Féraud, Georgios Z. Papadopoulos, and Patrick Maillé. 2018. Reinforcement Learning Techniques for

Optimized Channel Hopping in IEEE 802.15.4-TSCH Networks. In *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*.

[12] Wei Dong, Chun Chen, Xue Liu, Yuan He, Yunhao Liu, Jiajun Bu, and Xianghua Xu. 2014. Dynamic Packet Length Control in Wireless Sensor Networks. In *IEEE Transactions on Wireless Communications*, Vol. 13.

[13] Wei Dong, Jie Yu, and Pingxin Zhang. 2015. Exploiting Error Estimating Codes for Packet Length Adaptation in Low-Power Wireless Networks. In *IEEE Transactions on Mobile Computing*, Vol. 14.

[14] Lixin Duan, Ivor W. Tsang, Dong Xu, and Tat-Seng Chua. 2009. Domain Adaptation from Multiple Sources via Auxiliary Classifiers. In *Annual International Conference on Machine Learning*.

[15] Emerson. 2022. Emerson. https://www.emerson.com/en-us/expertise/automation/industrial-internet-things/pervasive-sensing-solutions/wireless-technology

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

[17] Paul Gardner, Xuanang Liu, and Keith Worden. 2020. On the Application of Domain Adaptation in Structural Health Monitoring. *Mechanical Systems and Signal Processing* 138 (2020), 106550.

[18] Mohsen Ghafoorian, Alireza Mehrtash, Tina Kapur, Nico Karssemeijer, Elena Marchiori, Mehran Pesteie, Charles R. G. Guttmann, Frank-Erik de Leeuw, Clare M. Tempany, Bram van Ginneken, Andriy Fedorov, Purang Abolmaesumi, Bram Platel, and William M. Wells. 2017. Transfer Learning for Domain Adaptation in MRI: Application in Brain Lesion Segmentation. In *Medical Image Computing and Computer Assisted Intervention*.

[19] Han Guo, Ramakanth Pasunuru, and Mohit Bansal. 2020. Multi-Source Domain Adaptation for Text Classification via DistanceNet-Bandits. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[20] IEC. 2017. WIA-FA. https://webstore.iec.ch/publication/32718

[21] IETF. 2020. 6TiSCH: IPv6 over the TSCH mode of IEEE 802.15.4e. https://datatracker.ietf.org/wg/6tisch/documents/

[22] ISA100. 2018. ISA100. https://isa100wci.org/

[23] Piumika N. Karunanayake, Andreas Könsgen, Thushara Weerawardane, and Anna Förster. 2023. Q learning based adaptive protocol parameters for WSNs. *Journal of Communications and Networks* 25 (2023). Issue 1.

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

[25] Brian Kulis, Kate Saenko, and Trevor Darrell. 2011. What You Saw Is Not What You Get: Domain Adaptation Using Asymmetric Kernel Transforms. In *IEEE Conference on Computer Vision and Pattern Recognition*.

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

[27] Hannes Larsson, Farnaz Moradi, Jalil Taghia, Xiaoyu Lan, and Andreas Johnsson. 2023. Domain Adaptation for Network Performance Modeling with and without Labeled Data. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*.

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

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

[30] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. 2015. Learning Transferable Features with Deep Adaptation Networks. In *Proceedings of the 32nd International Conference on Machine Learning*.

[31] Chenyang Lu, Abusayeed Saifullah, Bo Li, Mo Sha, Humberto Gonzalez, Dolvara Gunatilaka, Chengjie Wu, Lanshun Nie, and Yixin Chen. 2016. Real-Time Wireless Sensor-Actuator Networks for Industrial Cyber-Physical Systems. *Proc. IEEE* 104, 5 (2016), 1013–1024.

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

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

[34] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. 2009. Domain Adaptation with Multiple Sources. In *Advances in Neural Information Processing Systems*.

[35] James Manyika, Michael Chui, Jacques Bughin, Richard Dobbs, Peter Bisson, and Alex Marrs. 2013. Disruptive Technologies: Advances that will Transform Life, Business, and the Global Economy. http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/disruptive-technologies

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

[37] NSNAM. 2019. ns-3 Network Simulator. https://www.nsnam.org/

[38] OMNeT++. 2021. Source Code of OMNeT++. https://github.com/omnetpp/omnetpp

[39] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. 2021. A Survey of the Usages of Deep Learning for Natural Language Processing. *IEEE Transactions on Neural Networks and Learning Systems* 32, 2 (2021), 604–624.

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

[41] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. 2019. Moment Matching for Multi-Source Domain Adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

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

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

[44] Mo Sha. 2016. Testbed at the State University of New York at Binghamton. https://users.cs.fiu.edu/%7Emsha/testbed.htm

[45] Junyang Shi, Aitian Ma, Xia Cheng, Mo Sha, and Xi Peng. 2023. Adapting Wireless Network Configuration From Simulation to Reality via Deep Learning-Based Domain Adaptation. *IEEE/ACM Transactions on Networking* (2023), 1–16.

[46] Junyang Shi, Mo Sha, and Xi Peng. 2021. Adapting Wireless Mesh Network Configuration from Simulation to Reality via Deep Learning based Domain Adaptation. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

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

[48] Qian Sun, Rita Chattopadhyay, Sethuraman Panchanathan, and Jieping Ye. 2011. A Two-Stage Weighting Framework for Multi-Source Domain Adaptation. In *Advances in Neural Information Processing Systems*.

[49] Shiliang Sun, Honglei Shi, and Yuanbin Wu. 2015. A Survey of Multi-Source Domain Adaptation. *Information Fusion* 24 (2015), 84–92.

[50] TOSSIM. 2021. Source Code of TOSSIM. https://github.com/tinyos/tinyos-main/tree/master/tos/lib/tossim

[51] Hang Wang, Minghao Xu, Bingbing Ni, and Wenjun Zhang. 2020. Learning to Combine: Knowledge Aggregation for Multi-source Domain Adaptation. In *Computer Vision – ECCV*.

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

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

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

[55] WCPS. 2018. Wireless Cyber-Physical Simulator (WCPS). http://wsn.cse.wustl.edu/index.php/WCPS:_Wireless_Cyber-Physical_Simulator

[56] WirelessHART. 2024. WirelessHART. https://fieldcommgroup.org/technologies/wirelesshart

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

[58] Ruijia Xu, Ziliang Chen, Wangmeng Zuo, Junjie Yan, and Liang Lin. 2018. Deep Cocktail Network: Multi-Source Unsupervised Domain Adaptation with Category Shift. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

[61] Sicheng Zhao, Guangzhi Wang, Shanghang Zhang, Yang Gu, Yaxian Li, Zhichao Song, Pengfei Xu, Runbo Hu, Hua Chai, and Kurt Keutzer. 2020. Multi-Source Distilling Domain Adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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