Detecting Unseen Anomalies in Network Systems by Leveraging Neural Networks

Mohammad J. Hashemi, Eric Keller, Saeid Tizpaz-Niari

***Abstract*—Despite all the progress achieved in recent years in detecting anomalies in network systems, detecting unseen anomalies such as zero-day attacks still remained a challenging task. Traditional signature-based Network Intrusion Detection Systems (NIDS) cannot detect such anomalies as there exists no known signature for them. Moreover, Machine Learning-based (ML-based) NIDS trained with a vanilla supervised learning method cannot detect them as they come from a different distribution compared to what the model has been trained on. Domain adaptation techniques help transfer the knowledge gained from a labeled source domain to an unlabeled target domain. Such techniques have the potential to make a model trained on a dataset containing a few network attacks to detect new types of anomalies that might happen in the future. However, recent domain adaptation methods have been mostly designed for images and provide very limited benefits when applied to network traffic. In this paper, we introduce Proportional Progressive Pseudo-Labeling (PPPL), an effective approach for building a more general domain adaptation technique that can be leveraged to detect unseen anomalies in network systems. At the beginning of the training phase, PPPL progressively reduces target domain classification error, by training the model directly with pseudo-labeled target domain samples, while excluding samples with pseudo-labels that are more likely to be wrong from the training set and postponing training on such samples. Our evaluation conducted on the CICIDS2017 dataset shows that PPPL can significantly outperform other baselines in detecting unseen anomalies with up to 58% improvement based on the average F1 score.**

***Index Terms*—Anomaly detection, intrusion detection system, domain adaptation, deep learning, transfer learning, zero-day attacks.**

I. Introduction

The important role that computer networks play in our everyday lives provides a great incentive for attackers to interfere with them to take advantage of the available resources, access sensitive data, or deny legitimate users access to a specific service. The cost of these attacks on exploited businesses can be huge. Based on one estimation, the average cost of a denial of service (DoS) attack on a company was $1.7 million in 2018, and the average cost of all cybercrimes on each affected company was $13 million [1]. Based on another estimation, the annual cost of cybercrime on the whole world was estimated to be around $6 trillion in 2021 [2]. But the monetary cost for exploited businesses is not the only problem with these attacks. In fact, their impact can be much harsher, such as wide-scale power outages [3].

The severe adverse effects of these cyber-attacks have forced companies to invest billions of dollars in designing and building better tools to earlier detect and potentially prevent them [4]. Network Intrusion Detection Systems (NIDS) are one line of defense against such attacks. Despite all the progress made in designing better NIDS in recent years, they still struggle to detect zero-day attacks because they rely on signatures of *known* attacks. Zero-day attacks will bypass these signature-based NIDS as there exists no known signature for these attacks.

An alternative approach is to utilize a machine learning model to detect these new and unseen anomalies. Different unsupervised learning methods have been proposed to tackle this problem in the past. However, models trained with these methods can only detect different network attacks while generating a high number of false alerts or false positives, which happen when the model classifies traffic as malicious, but it is in fact benign. Any attempt to bring down their false-positive rates significantly deteriorates their ability to detect real malicious traffic. But, a good NIDS should be able to detect a variety of network attacks while keeping false alerts at a low level, because a high false-positive rate significantly increases the workload of security experts in sorting through all the traffic labeled as malicious to identify the real attacks.

Supervised learning can also be used, where one labels a portion of network traffic that also includes some of the known attacks and trains a model using this dataset in a supervised fashion to detect anomalies in the future. However, while such a model works well in detecting network attacks that are included in the training set, it does not work well in detecting unseen anomalies that might happen in the future. This could happen with unseen attacks, whose characteristics are different, though likely similar, from those the model is trained on them. It could also happen because the input data distribution can constantly change, and we might observe a domain gap between the current data distribution and that which was used to train our model. In such cases where we observe a domain shift in input data, the model cannot do a good job of detecting different types of anomalies. [5], [6].

In this paper, we try to answer the following question: How can we train a model to detect unseen network anomalies that might have different characteristics compared to those we used to train a model? Unsupervised domain adaptation methods are used to address such problems where there is a domain shift between data distributions of a labeled source domain and an unlabeled target domain. Recently, many different domain adaptation methods have been proposed [7]–[17].

Mohammad Hashemi and Eric Keller are with the Department of Electrical, Computer, and Energy Engineering, University of Colorado Boulder, Boulder, CO, 80309 USA (e-mail: mohammad.hashemi@colorado.edu; eric.keller@colorado.edu). Saeid Tizpaz-Niari is with the Department of Computer Science, University of Texas at El Paso, El Paso, TX, 79968 USA (e-mail: saeid@utep.edu)

Despite different techniques used in recent approaches, one common flaw among them is that they do not generalize well across different types of inputs - in particular, network traffic. Some of the methods, such as [9], [12], are intrinsically designed for images, as they do image-to-image translation at the pixel level or need specific data augmentation that should be applied to them at the pixel level. Therefore, there is no straightforward way to apply these techniques to other input types such as network traffic. For the other approaches, they either leverage adversarial loss [8], [11], [18] or other techniques such as clustering [7]. One common problem with those approaches is that there is no guarantee of preventing the wrong alignment of the target samples. In other words, target domain representations from one class can get aligned with another class during the domain adaptation, leading to lower performance of the model. In addition, the complexity and the large number of hyperparameters that some of these methods have weighs on this problem as there is no straightforward way to find the hyperparameters that minimize the target error due to the lack of a labeled validation set for the target domain. Therefore, more complexity leads to less generalization.

In this paper, we introduce Proportional Progressive Pseudo-Labeling (PPPL), a more general domain adaptation technique that works across different input types. PPPL assigns pseudo-labels to the target samples and trains the model directly with them. Key to PPPL is that it tries to minimize the number of target samples that will align with a wrong class by excluding uncertain samples from the training set at the beginning of the training procedure and progressively bringing them back into the training loop with a weight proportional to their certainty. For this, we assume that we can guess the proportions of target samples that belong to each class. Note that while we do not know the individual labels in the target domain, the class proportions can be guessed in many cases. We consider three different scenarios with regard to this condition. In fact, this paper is an extension to our conference paper [19] in which we assumed that target domain class proportions can be guessed accurately. In this paper, in addition to that scenario, we evaluate our method in a scenario in which we consider some different levels of error in the ratio of malicious traffic in the target domain. Moreover, we further relax this assumption and discuss how PPPL can be modified to work in a more realistic scenario in which there is no information about the target domain class proportions.

Our experiments on the CICIDS2017 [20] network intrusion-detection dataset demonstrate that our approach is superior to other baselines for the anomaly detection task in network traffic. Our evaluation shows that PPPL significantly outperforms other baselines in detecting unseen anomalies with up to 58% improvement based on the average F1 score.

II. Background

In general, unsupervised domain adaptation is the process of mitigating the domain shift between a labeled source dataset and an unlabeled target dataset to transfer the knowledge gained from the source dataset to the target dataset. These techniques are leveraged during the training of a model and the goal is to make a model predict target samples more accurately without directly labeling them. For example, domain adaptation techniques can enable an image classifier that was trained on labeled images with white backgrounds to more accurately classify unlabeled images that are captured with a webcam from the objects in an ordinary office [7], [8]. Recent methods that have been proposed for domain adaptation leverage a wide range of techniques to make a classifier achieve better results on the target domain. Some of them, such as [9], [12], are explicitly designed for images. Hoffman et al. [12] introduced CyCADA, an image-to-image translation method in which they combined generative adversarial networks with cycle-consistency constraints at the pixel level and semantic-consistency constraints at the feature level to reduce the domain shift between the source and target domains. French et al. [9] built their domain adaptation method on top of the mean teacher [21] idea and called it self-ensembling (SE). Basically, in addition to training a model with source samples and cross-entropy loss, two different stochastic transformed versions of each target sample are fed to two different models called student and teacher, and the squared difference of their outputs is minimized. As French et al. described in their paper, for some datasets, the transformations they used for their stochastic data augmentation are highly tuned towards that dataset and therefore, cannot be easily applied to other input types. This group of works is intrinsically designed for images and cannot be applied to other input types in a straightforward manner.

The second group of methods can be applied to different input types as their focus is to mitigate the gap between the source and the target domains in the feature space at some intermediate layer of a deep network. The adversarial domain adaptation is the basic idea behind a large portion of recent approaches [8], [11], [18]. In these works, the classifier is trained jointly with a domain discriminator like a GAN [22]. The discriminator is trained to distinguish between source and target samples based on their representation captured from an intermediate layer of a deep network, while the classifier itself is trained in a way to fool the discriminator that results in generating domain invariant features. Ganin et al. [11] proposed Domain Adversarial Neural Network (DANN) in which they augment a classifier with a domain discriminator and train them in an adversarial fashion by back-propagating the reverse gradients of the domain classifier to learn domain invariant representations. Long et al. [8] designed Conditional Adversarial Domain Adaptation (CDAN) in which they condition the domain discriminator on the cross-covariance of domain-specific feature representations and classifier predictions, as well as on the uncertainty of the classifier to prioritize the discriminator on the easy to transfer samples. While we also model target domain uncertainty into our PPPL method, our approach differs from approaches like CDAN as they model the uncertainties into the domain discriminator. Doing so makes those approaches deal with the complications of training GANs, whereas in our approach there is no discriminator and we model uncertainty directly into the classification loss. This results in less complexity and greater generalization.

Beyond adversarial domain adaptation methods, there are also other approaches like Contrastive Adaptation Network (CAN). Kang et al. [7] mitigate the gap between the source and target domains at some feature space through an alternating optimization method in which they initially cluster target samples into multiple different groups with some complicated clustering method, and then they assign some pseudo-labels to them. They then train the model by minimizing intra-class discrepancy and maximizing inter-class discrepancy. Similar to CAN, we also assign some pseudo-labels to the target samples, but unlike CAN, we do not use any clustering method. We instead assign the pseudo-labels directly based on the model predictions. This, again, leads to less complexity and greater generalization. As we will show in the evaluation section, while these domain adaptation methods perform well in the image classification task, they do not provide the same level of benefit for detecting unseen anomalies in network traffic.

Therefore, our method differs from other domain adaptation methods in two major ways: one important difference is that PPPL reduces the domain gap between the source domain and the target domain at the final layer of the network (logit space), whereas other domain adaptation methods reduce domain gap at input space (such as in CyCADA) or at some intermediate representation of the inputs (such as in CDAN, DANN, etc.). Another major difference between PPPL and other domain adaptation methods is that when using other methods, in addition to training the main model, some extra components should also be trained. For example, in adversarial domain adaptation methods such as CDAN and DANN, a domain discriminator also gets trained jointly with the main model, or in CAN, we need to alternate between training the main model and optimizing a clustering algorithm, constantly. In contrast, in PPPL we only train the main model and no extra optimization step or extra model is required. This reduces the number of hyperparameters used in PPPL compared to other methods and makes it a more suitable unsupervised domain adaptation method.

Semi-supervised learning (SSL) methods [21], [23]–[27] are also used to train a model with a combination of labeled and unlabeled samples and have the potential to be used for detecting unseen anomalies in the network traffic. Many recently proposed SSL methods add a loss term, calculated based on the unlabeled data to make the model generalize better to unseen data. As discussed by Berthelot et al. [28], these SSL methods can be categorized into three different classes. Some of these techniques [23], [24] aim to minimize the entropy of the model's outputs on unlabeled data to make the model output more confident predictions on the unlabeled data. The second class of SSL methods [21], [25], [26] regularizes the model to have consistent outputs when provided with different perturbed versions of the same unlabeled input. Finally, the last class of SSL methods [27] aims to make the model generalize better by leveraging some generic regularization techniques to prevent the model from overfitting the training data. MixMatch unifies all of these techniques to benefit from all of them. In MixMatch, first, the model will be provided with K random perturbed versions of each unlabeled sample, and a unique probability vector will be "guessed" and assigned to all of those K inputs to achieve consistency regularization. Then, this probability vector will be sharpened by adjusting the "temperature" of this categorical distribution to minimize its entropy. Finally, to achieve generic regularization, for each batch of the labeled and unlabeled data points, a new batch will be created in which each sample is built from a linear combination of two random samples from the original batch. In the end, the model will be trained with the labeled samples by using cross-entropy loss and with the unlabeled ones by minimizing the L2 norm of model outputs and the "guessed" labels. While we also assign some pseudo-labels to the unlabeled samples during our training procedure, we do not mix up the samples. Also in contrast to MixMatch, we model the certainty of our guessed labels into the training procedure and postpone the training on the less certain samples. Furthermore, unlike MixMatch we directly reduce the domain gap between labeled and unlabeled samples in the logits space and have fewer hyperparameters. As we will show, all of these differences make PPPL generalize better than MixMatch and get better results on the anomaly detection task in the network traffic.

III. Design Insights

Our goal is to train a neural network with the help of a labeled source domain in order to detect unseen anomalies in an unlabeled target domain. In other words, given a set of labeled samples known as source domain $S = \{(x^s_1, y^s_1), (x^s_2, y^s_2), ..., (x^s_{N_s}, y^s_{N_s})\}$ such that $y_i \in Y = \{0, 1\}$ and another set of unlabeled samples known as target domain $T = \{x^t_1, x^t_2, ..., x^t_{N_t}\}$ which come from two different data distributions our goal is to train a model $F : x \mapsto y$ to predict $\hat{y}^t_i \in Y$ to maximize the F1-score on the target domain. In doing so, we essentially can label a portion of network traffic containing a few network attacks and be able to detect new types of anomalies in the rest of the unlabeled traffic with the help of our approach. In the rest of this section, we provide some insights that helped us to design our approach. Then, in the following section, we explain how we incorporate these insights to design PPPL.

A. Insight 1 - when training with pseudo-labels mean square error is a better choice

As we mentioned earlier, we want to assign pseudo-labels directly based on model predictions to the target domain samples and train the model with them. For training, there are two choices here: We can feed the outputs of the final layer of the model to a Softmax function and train the model with the cross-entropy (CE) loss. We can also directly minimize the distance of the final layer outputs and the one-hot encoding of the labels with mean square error loss (MSE). We argue that for such a setting, MSE is a better choice. First, consider the Softmax function, which is defined as follows:

$$\sigma(z)_i = \frac{e^{z_i}}{\Sigma_{j=0}^{M-1} e^{z_j}} \text{ for } i = 0, 1, ..., M-1$$

where M is the number of classes. Note that because of the nature of this function, there is no one-to-one mapping between the probabilities (outputs of the Softmax layer) and the logits (inputs of the Softmax layer). That is to say, many different points in the logit space can be mapped to a single vector of probabilities. For example, when there are only 2 classes,

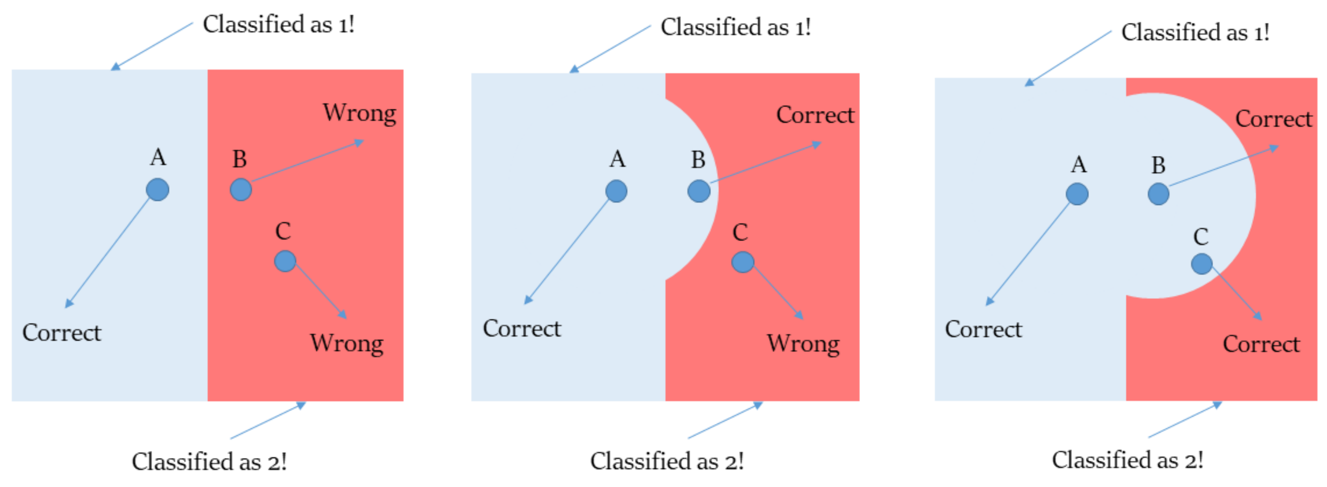


Fig. 1: The illustration of training a model on the correct target domain samples. In (a) there are three target domain samples, and only point A is predicted correctly. When we train the model on point A and label it as 1, the model learns that close proximity of A should also be classified as class 1 as illustrated in (b) and therefore, point B also will be classified correctly. Then this effect gets propagated to the points near B and therefore point C will be classified correctly in the next iteration as illustrated in (c).

both of the points $[1, 100]$ and $[200, 299]$ will be mapped to the $[\frac{1}{1+e^{99}}, \frac{e^{99}}{1+e^{99}}]$. This means that potentially, the points that belong to the same class can form multiple different clusters in the logit space. More specifically, the target domain samples and the source domain samples that share the same label or pseudo-label can fall into different clusters. On the other hand, when we train the model with MSE, the points that have real or pseudo-label C_i will fall into one single cluster very close to the point $[0, ..., 0, 1, 0, ...0]$ where the i-th index is 1 after we train the model on them. This characteristic is more desirable as it mitigates the domain gap between the source and the target samples at the logit space and forces the model to learn features in the earlier layers of the network that leads to indistinguishable representations at the logit space between the source and the target domain samples. From another point of view, if we had a domain discriminator to distinguish between the source and the target samples' logits it could be completely fooled. Therefore by using MSE loss in this setting, we get the same advantages of adversarial domain adaptation techniques without being worried about the complications of training GANs.

B. Insight 2 - training only on correct samples gradually reduces the target error

Consider a model pre-trained on the source domain. We argue that we can make the model predict more target domain samples correctly if we further train this model on the target domain samples that are already classified correctly. Also, we argue that if we do this technique for more iterations, we can gradually reduce the target error further.

To understand this better, consider a model pre-trained on the source domain. Also, as illustrated in Figure 1 consider some samples (e.g., A, B and C) from the target domain that belongs to the same class (e.g., class 1) and fall into close proximity of each other at some middle representation of the network. Assume that we assign pseudo-labels to these samples directly based on model predictions. Suppose that some of these pseudo-labels are correct and some are wrong (e.g. $\hat{y}_A = 1, \hat{y}_B = 2, \hat{y}_C = 2$). If we exclude wrong samples (B, C) and train the model only on correct samples (A), then the model learns that the points (B) near these points (A) are also more likely to be from the same class (class 1) and potentially some of them will get a correct pseudo-label in the next iteration. This effect gets propagated to the points near B (C) in the next iteration. Therefore, excluding wrong samples and training only on correct samples gradually reduces the target error.

To further show this, in Figure 2 we demonstrate the results of such training on three different domain adaptation tasks from the CICIDS2017 dataset. This dataset is discussed in more detail in Section V-A. For each of these tasks, the model was trained on a source domain that includes some of the network attacks and the results are reported on a target domain that consists of different types of network attacks. As can be seen in all cases, the model progressively learns to detect unseen anomalies in the target domain better. Note that F1 scores keep increasing during the first few epochs for all three cases. After 10 epochs, the results achieved with this method are better than the results of any other domain adaptation approach that we are aware of by a large margin. In other words, we can significantly reduce the target error only by assigning pseudo-labels to the

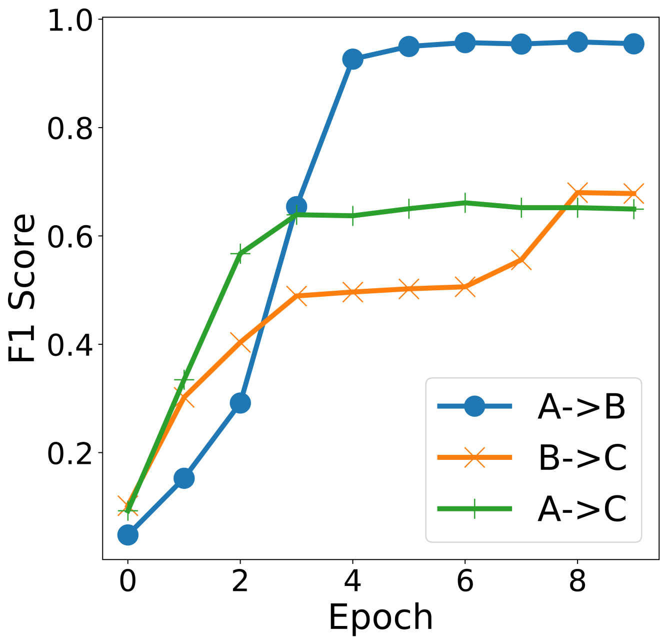


Fig. 2: F1 scores reported for the target domain when trained only on the correct predictions for three different domain adaptation tasks.

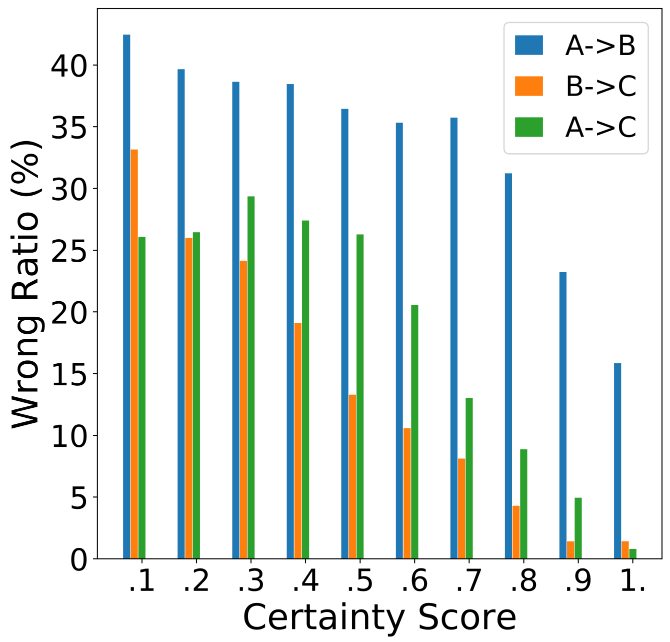


Fig. 3: The ratio of wrong predictions at different levels of model certainty.

target samples directly based on the model predictions and training the classifier with them. We just somehow need to determine in which cases the model is wrong to exclude them from the training procedure.

C. Insight 3 - an uncertainty metric can guide which predictions are wrong

Unfortunately, there is no straightforward way to know which of the model predictions are correct and which ones are wrong on the target domain as we do not know the target domain's labels. But, among all samples that are predicted as the same class, there is a relation between the model's certainty and the chance of wrong predictions. We capture the model's certainty with the difference between the two scores that the model outputs for each sample and call it the certainty score. This difference becomes smaller as in some intermediate representations of the inputs the points get further away from the same-labeled source points, falling into subspaces that are not well explored by the model or when they fall in close proximity to other points with different labels meaning getting closer to the decision boundaries. Thus, in such cases, it becomes more likely to get predicted wrongly.

This characteristic can also be seen in Figure 3. In general when the certainty score decreases among samples that are given the same pseudo-label, a larger portion of predictions becomes wrong. For this figure, we first trained the model on the source domain for each of the aforementioned tasks with MSE loss. Then, the ratio of wrong predictions to all of the target samples that are predicted as benign and their certainty scores fall into the interval $[\frac{i-1}{10}, \frac{i}{10}]$ is calculated.

D. Insight 4 - the timing of inclusion of a wrong-prediction matters

While we can predict better, we cannot know for sure which predictions are correct, and therefore it might be inevitable we assign some wrong pseudo-labels to some of the target samples and train the model on them. But one thing that is important is the time when we train the model on the target samples with the wrong pseudo-labels. We argue that the early inclusion of such samples into the training procedure deteriorates the model's performance on the target domain more than later inclusion. This is because of the same phenomenon that we discussed in insight 2: When we train a model on a target sample with a wrong pseudo-label, it would be more likely for the model to assign that wrong label to the points that are in close proximity of that wrong sample. Then, this wrong label propagates to the neighborhood of these newly affected samples in the next iteration. Therefore, the earlier we train the model on a wrong sample, the further its impact will propagate, the more the model's performance deteriorates on the target domain.

This problem can also be seen in Figure 4. For this figure, for each of the tasks mentioned in the second insight, we trained the model the same way we discussed but we also included some samples with wrong pseudo-labels into the training procedure at different epochs (epochs 1,4,7, and 10). The size of the wrong pseudo-labeled samples in each task is set to be at most 10% of the target domain sample size. For all of the cases, we trained the model for ten epochs. In this figure, we illustrate the change in the F1 score of the target domain when the wrong pseudo-labeled samples were included in the training loop at epochs 1, 4, and 7 in comparison with when they were included at epoch 10. Therefore a larger bar shows a greater decrease in the F1 score. As can be seen, the

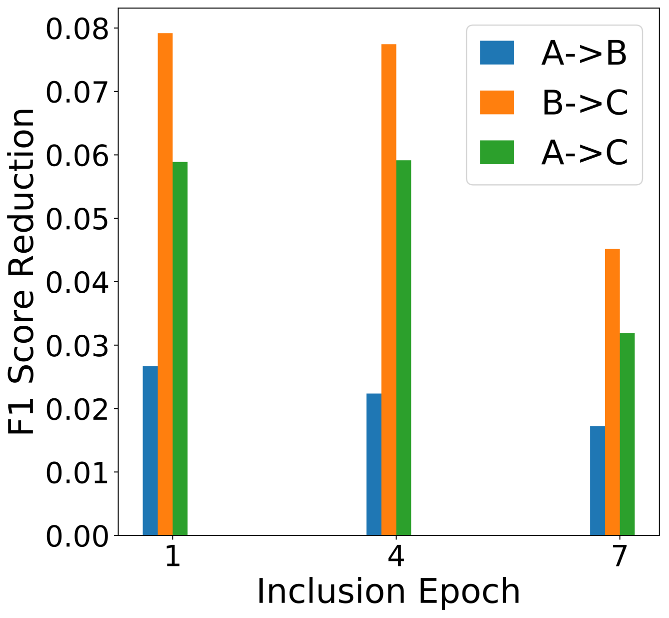


Fig. 4: The negative impact of early training on wrong pseudo-labels.

earlier the model gets trained on the wrong samples, the more its ability to detect unseen anomalies deteriorates. For example, for the B $\rightarrow$ C task, if we include the wrong pseudo-labeled samples at the first epoch of training, the final F1 score will be almost 8% lower than when we postpone their inclusion to epoch 10.

IV. Proportional Progressive Pseudo-Labeling (PPPL)

In this section, we first demonstrate our method for a scenario in which target domain class proportions are known to us. Then we discuss how this method can be extended to work for scenarios in which such information is not available.

A. PPPL with Prior Knowledge about Target Domain Class Proportions

Based on the insights we discussed, we designed our approach. In a nutshell, based on the second insight, we know that if we use a model pre-trained on the source domain, assign pseudo-labels to the target samples with it and exclude the wrong pseudo-labels the model progressively gets better. Also, based on the first insight, we know that using MSE loss is better than CE loss for such a setting. Unfortunately, we cannot find out exactly which pseudo-labels are wrong since we do not know the target labels. However, based on the third insight, we know that the ratio of wrong predictions increases as the model certainty decreases. In addition, based on the fourth insight, we know that it is better to postpone training the model on such samples.

Algorithm 1 describes our approach in more detail. The inputs are F, which is the model pre-trained on the source domain, X_s, which is the set of all source domain samples, Y_s, which is the set of all source domain labels, X_t, which is the set of all target domain samples and CP_t, which is the set of target class proportions that are guessed or known from other sources. We first train the model (F) on the source domain samples (X_s, Y_s) with the MSE loss function (based on insight 1). Then in each iteration of the algorithm, we first get the score which is a vector with size 2 (2 is the number of available classes) for each of the target samples (line 4) and assign a pseudo-label to that sample based on its largest score (line 5). Then, for all of the target samples, we calculate the "certainty score" (line 6) and then assign a weight value to each of the target samples based on its certainty score (line 7). This weight will be used later during training to control the impact of each sample on the model parameters.

Algorithm 1 Proportional Progressive Pseudo-Labeling

```
1:  procedure PPPL($F, X_s, Y_s, X_t, CP_t$)
2:      for $i \leftarrow 1$ to 45 do
3:          $N \leftarrow 10 + 2 \times i$
4:          $S_t \leftarrow F(X_t)$
5:          $PL_t \leftarrow argmax(S_t)$
6:          $CS_t \leftarrow CalcCertaintyScore(S_t)$
7:          $W_t \leftarrow CalculateWeight(CS_t, PL_t, N)$
8:          $X'_t, Y'_t, W'_t \leftarrow Adjust(X_t, PL_t, W_t, CP_t)$
9:          $X'_s, Y'_s, W_s \leftarrow Select(X_s, Y_s)$
10:         $Train(F, X'_s, Y'_s, W_s, X'_t, Y'_t, W'_t)$
11:     end for
12: end procedure
```

The function $CalculateWeight(CS_t, PL_t, N)$, first groups all of the samples that are assigned the same label. Then, for each group, it assigns a weight between $[0.2-1.0]$ to $N\%$ of the samples and 0 to the rest of the samples of that group. Within each group, the weights are monotonically assigned based on the certainty scores such that a sample with a larger certainty score will be assigned a larger weight. More specifically, if the number of samples that fall into the top $N\%$ for a given group is L_c then the weights for those samples are calculated as follows: $w_j = \frac{1}{t_j}$ where $t_j = 1 + \frac{4}{L_c} \times j$ in which $j \in [0, L_c)$ and w_j is assigned to the j-th sample with the largest certainty score.

For better illustration, in Figure 5, we show the weights that will be assigned to the target samples with the same pseudo-label at epochs 1, 20, and 45 of our method. We assumed that the size of this group would remain at 1000 during the whole training. As can be seen, a non-zero weight will be assigned to only 10% of the samples, during the first epoch. This essentially means that the other 90% of the samples that have a lower certainty score will be excluded from training in the first epoch. Also, note that we include more samples in the training data in the later stages of our approach. For example, 50% of the samples will be included in the training data, at epoch 20, and at the final epoch, all the samples will be included. Using this weighting strategy decreases the chance of training on wrong samples in the early epochs by excluding the less certain samples (designed based on insights 3 and 4). Also, for the samples that will be included at each epoch, we assign a smaller weight to the less certain samples to decrease

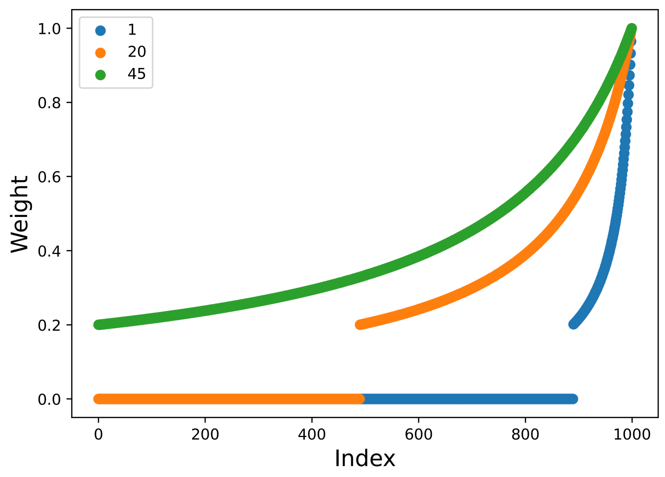


Fig. 5: The illustration of weights assigned to the target samples with the same pseudo-label.

the impact of those that are assigned wrong pseudo-labels and potentially are among the included samples on the model parameters (designed based on insight 3).

There is also another way we try to reduce the number of wrong samples which is based on guessed class proportions (line 8). In the $Adjust$ function, for each label, we first calculate the ratio of samples with that pseudo-label to the whole target domain sample size. Then, during the first 30 iterations of our algorithm, we change the pseudo-labels of the most uncertain samples of the class whose ratio is higher than its corresponding guessed ratio to the opposite class. Because it means that the model predicted samples with that label more than what we expected. For cases where the target dataset is very imbalanced, this technique helps the model detect anomalies better in the early iterations of the algorithm. For the rest of the iterations, we train the model with the exclusion of the least certain samples which are predicted more than their expected class proportions. Because we expect some of these predictions to be wrong. Therefore, from the class that is predicted more than its expected value, we keep excluding the most uncertain predictions until the ratio of remaining samples becomes equal to our expected ratio (designed based on insights 2 and 3).

In addition to the target samples, we also select some samples randomly from the source domain at each epoch to train the model on them. We do this because in the initial iterations of our algorithm we only include a small portion of the target samples, and we do not want to make the model overfit to them. We also assign a weight equal to 1 to these source samples as all of their labels are correct. Finally, with the combination of pseudo-labeled target samples and these source samples, we train the model with MSE loss (designed based on insight 1). More specifically, the loss function we use for training is as follows:

$$Loss_{PPPL} = \frac{1}{N}\sum_{i=1}^{N} w_i||F(x_i) - y_i||_2^2$$

in which y_i is the one-hot encoding of (pseudo)label assigned to sample x_i and N is the total number of included samples at the current epoch.

B. PPPL with No Prior Knowledge about Target Domain Class Proportions

While having knowledge about target domain class proportions can help to better detect unseen anomalies in network systems, PPPL can also be used in more realistic scenarios in which such knowledge does not exist in advance.

In this situation, instead of fixed class proportions, we run our algorithm by enforcing adaptive class proportions as follows:

$$CP_t = (1-w).CP_{t-1} + w.\hat{CP}_t$$

$$CP_0 = CP_{src}$$

where CP_{src} is the source domain class proportions, $\hat{CP}$ is the class proportions calculated based on the pseudo-labels assigned to the target domain samples, and w is a hyperparameter in the range of $[0,1]$. In other words, during the first iteration of our algorithm, we enforce source domain class proportions. Note that the source domain class proportions can always be calculated as we know the source domain labels. Then, during the next iterations, we update these values using an exponential moving average of the *predicted* class proportions of the target domain (which are calculated based on their assigned pseudo-labels).

In this formula, w controls how strongly we want to enforce source domain class proportions: When w is 0, we enforce the source domain class proportions during all the iterations of our algorithm, which will be beneficial for the cases where the source domain class proportions are very similar to the real target domain class proportions. On the other hand, as w grows, we make our estimation to be more similar to the latest predicted target domain class proportions, which might be more beneficial for the cases where source domain and target domain class proportions are different. Therefore, different values of w lead to models which have different detection rates. Since we do not know which w works the best, in order to detect unseen anomalies with PPPL in this scenario, we train an ensemble of models, each using a different w and average their predictions to calculate our final score as follows:

$$FinalScore = \frac{1}{K}\sum_{i=1}^{K} F_i(x)$$

where K is the total number of models in our ensemble, each of F_i is a model trained with a different value of w, and $F_i(x)$ is the output of a model, which is a $2-$dimensional vector.

If we could find a way to select the model which detects anomalies in the target domain better in the ensemble of our models, then the average detection rate could be further increased. Note that since we do not know the labels of the target domain samples, there is no straightforward way to select the best model. However, we can approximate the detection rate in the target domain with the help of source domain samples. Among the models that we train, we expect the one that detects different perturbed versions of anomalies in the source domain with a higher rate to generalize better and therefore have a higher chance of detecting anomalies in the target domain. In practice, we defined a new metric to measure the generalization capacity of each of the models in our ensemble as follows:

$$GeneralizationScore = \sum_{i=1}^{20} F1(X_{src} + 0.01 \times i \times R, Y_{src})$$

$$R \sim Uniform(0,1), X \in [0,1]$$

where R is a random matrix with the same size as X_{src} which is generated from a Uniform distribution, and $F1(.)$ is the F1 score calculated for a given model when provided by a perturbed version of the source domain samples. In other words, the generalization score of a model is the summation of the F1 scores calculated on the source domain when perturbed with different levels of noise and a higher value shows better generalization capacity. Therefore, in our second approach, instead of getting the average of the models' outputs in our ensemble, we select the one with a higher generalization score and only use that model to detect unseen anomalies in the target domain. In the evaluation section, we show how using these two techniques can help us to relax our assumption about target domain class proportions and make our method work in a more realistic scenario.

V. Evaluation

In this section, we evaluate PPPL's ability to detect unseen anomalies in network traffic. We first demonstrate how well PPPL can perform when the target domain class proportions can be guessed accurately, and then we show how this assumption can be relaxed.

A. Dataset

To evaluate how well PPPL can detect unseen anomalies in network traffic, we used the CICIDS2017 dataset [20]. We built three different subsets from this dataset such that each one consists of different types of network attacks as follows:

- **Domain A:** FTP-Patator, SSH-Patator.
- **Domain B:** DoS Slowloris, DoS Slowhttptest, DoS Hulk, DoS GoldenEye, Heartbleed.
- **Domain C:** Web attacks, Infiltration.

We consider the traffic collected in each of these datasets as one domain and define 6 domain adaptation tasks between each pair of them as follows: A→B, A→C, B→A, B→C, C→A, and C→B. In order to classify the packets in this dataset, we first preprocessed them with the same method described by Hashemi et al. in [29]. More specifically, we first grouped packets based on their source and destination IPs. Then, from each packet, we extracted some features from its Ethernet header, IP header, TCP header, and UDP header. We also extracted the inter-arrival time (the time between the current packet and the previous one in the same group), and direction (whether the packet is from the sender or receiver) of each packet. In total, we extracted 29 features for each packet. Finally, within each group, we concatenated the feature vectors of every 20 consecutive packets together to feed them to the classifier. Given this preprocessing method, the A, B, and C domains contain 573,544, 685,241, and 462,031 samples, respectively. Also, all of these domains are imbalanced. That is to say, a large portion of the packets in each of them are benign, and only a small portion is malicious. More specifically, the percentage of malicious traffic in the A, B, and C domains are 2.2%, 18.2%, and 2.7%, respectively. For evaluation we report the F1 score which equals to $\frac{TP}{TP+0.5(FP+FN)}$. We decided to use this metric because true negatives (benign samples that are predicted as benign) are not considered in the F1 score, which makes it a good metric for scenarios where we have imbalanced datasets, such as anomaly detection in network traffic. In addition, both false positives (benign samples that are predicted as malicious) and false negatives (malicious samples that are predicted as benign) equally affect the F1 score. Therefore, when a model achieves a higher F1 score, it means that its total wrong predictions (FP+FN) are fewer than the other ones.

B. Baselines

We compare PPPL with three different baselines: CAN [7], CDAN [8] and MixMatch [28]. CAN and CDAN are among the best domain adaptation techniques to the best of our knowledge, and MixMatch is one of the best semi-supervised learning methods, which has been recently published. More specifically, CAN is state-of-the-art for the Office-31 [30] dataset. Kang et al. in their evaluation showed that CAN outperforms many other baselines such as Domain Adversarial Neural Network (DANN) [11], [18], Joint Adaptation Network (JAN) [13], Multi-adversarial Domain Adaptation (MADA) [17], Deep Adaptation Network (DAN) [31] and Self Ensembling (SE) [9]. Also, CDAN is among the best adversarial domain adaptation techniques that we are aware of. Long et al. in their evaluation showed that CDAN outperforms many other methods such as DAN [31], Residual Transfer Networks (RTN) [32], DANN [11], [18], Adversarial Discriminative Domain Adaptation (ADDA) [33], JAN [13] and CyCADA [12]. Finally, Berthelot et al. [28] showed that MixMatch can outperform several other semi-supervised learning methods such as Π-Model [25], [26], Mean Teacher [21], Virtual Adversarial Training [34], Pseudo-Label [24] and MixUp [35] on multiple datasets. Therefore by comparing our approach with CAN, CDAN, and MixMatch and outperforming them, the superiority of PPPL over many other baselines can be inferred.

C. Implementation Details

To evaluate PPPL on the CICIDS2017 dataset, we implemented our method in the TensorFlow framework and run our experiments on a system with 32 GB of RAM, a 3.6Ghz Core-i7 CPU, and an Nvidia TITAN V GPU. For each of the six domain adaptation tasks we defined, we first trained a two-layer fully-connected network $(2048 \rightarrow 2)$ with ReLU non-linearity after the first layer on the source domain. We decided to use a simple model architecture with only one hidden layer as such a model does not need to make lots of computation. Therefore it can process network traffic faster than a model with multiple hidden layers. During this phase, we trained the model for six epochs with a batch size of 256 using Adam [36] optimizer and adjusted the learning rate by $\eta_p = \eta_0(1+\alpha)^{-\beta}$ where $\eta_0 = 0.0001, \alpha = 0.001, \beta = 0.75$ and $p = i/5$ where i is

TABLE I: Results of all methods on the CICIDS2017 dataset. The numbers reported in this table are F1 scores.

Method	A → B	A → C	B → A	B → C	C → A	C → B	Avg
Only-Src	0.055	0.215	0.028	0.087	0.010	0.016	0.068
CDAN	0.007	0.004	0.006	0.013	0.000	0.025	0.009
CAN	0.662	0.169	0.123	0.333	0.000	0.005	0.215
MixMatch	0.000	0.000	0.000	**0.708**	0.000	0.650	0.226
PPPL	**0.967**	**0.634**	**0.821**	0.661	**0.818**	**0.956**	**0.810**

the number of iterations passed from the beginning of the training. Since this dataset is imbalanced, in each epoch, we first downsampled the benign inputs to become the same size as malicious inputs and then trained the model on them. For the domain adaptation phase, we trained the model using Alg. 1. During this phase, we also used the Adam optimizer with a learning rate of 0.0001 and a batch size of 256, and also, similar to the initial phase, at the beginning of each epoch of our algorithm, we balanced our dataset. [1]

To evaluate other baselines, we used the codes published by the authors of those methods and trained the same model on each of the domain adaptation tasks we defined. We evaluated these methods on our network traffic dataset with several different hyperparameters and reported the best result they could achieve.

D. Scenario 1: Perfect Knowledge about Target Domain Class Proportions

To see how well PPPL can detect unseen anomalies in the network traffic, we first consider a scenario in which the target domain class proportions can be accurately guessed. Table I shows how PPPL performs in this situation in comparison to other baselines. As can be seen, our baselines perform very poorly on this dataset. In contrast, PPPL significantly improves the detection rate of unseen anomalies compared to only training on the source domain in all of the domain adaptation tasks, and also it is 58.4% better than the best baseline we compared with based on the average F1 score.

E. Scenario 2: Inaccurate Knowledge about Target Domain Class Proportions

Since it might not always be feasible to accurately guess the ratio of malicious traffic, in the second scenario, we evaluate our approach in a situation where some error exists in the guessed target domain class proportions. More specifically, for each of the domain adaptation tasks in this dataset, we modified the anomalous class proportion such that $|\hat{CP}_a - CP_a| = E \times CP_a$. where $E \in \{0.1, 0.2, ..., 0.7\}$, CP_a is the real ratio of the malicious traffic in the target domain, and $\hat{CP}_a$ is the guessed one.

Figure 6 demonstrates how our method performs in such a scenario. In this figure, the leftmost bar shows the average F1 score across all six domain adaptation tasks when there is no error in the guessed class proportions, and the other bars show the same metric while we considered different levels of

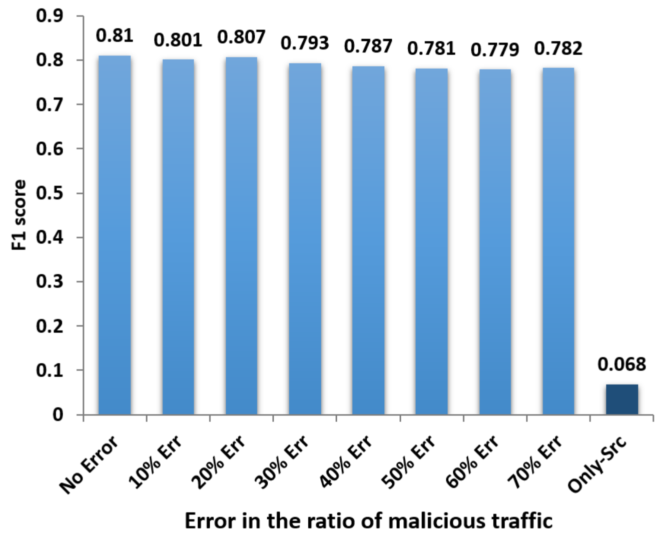


Fig. 6: Average F1 score across all the six domain adaptation tasks defined on the CICIDS2017 dataset when the guessed ratio of malicious traffic used in the PPPL algorithm has some error.

error. The right-most bar also shows the detection rate in the target domain when we only train the model on the source domain. Note that even when there is a 70% error in the guessed malicious traffic ratio, the detection rate only drops by 3% compared to the case when the target domain class proportions are guessed accurately. Also, note that it is still 71% better compared to the case where we only trained the model on the source domain samples. Therefore, in a scenario where the target domain class proportions can be guessed but with some error, our method still performs significantly better than other baselines in detecting anomalies in network traffic. More details on how PPPL performs on each individual task in this scenario can be found in Table IV in the appendix.

F. Scenario 3: No Prior Knowledge about Target Domain Class Proportions

In this scenario, we want to further relax the condition related to the target domain class proportions and see how well PPPL can detect anomalies when there is no prior knowledge about the target domain class proportions with the help of the ensemble method we described in Section IV-B.

In practice, for each of our domain adaptation tasks, we trained two models with $w = 0.1$ and $w = 0.7$. Figure 7 shows how PPPL in this scenario performs compared to the case when we have accurate knowledge about target domain

[1]For more details, please see https://github.com/s-mohammad-hashemi/AD_PPPL.

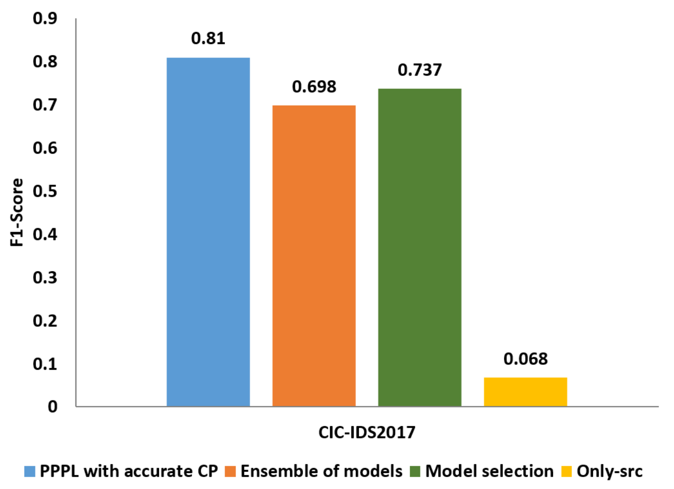


Fig. 7: Average F1 score across all the six domain adaptation tasks defined on the CICIDS2017 dataset when there exists no prior knowledge about target domain class proportions.

class proportions. In this figure, each bar is the average F1 score across all the six domain adaptation tasks in the CICIDS2017 dataset. Note that while the detection rate drops compared to the case where we know the target domain class proportions accurately, the average F1 score of our method in this scenario is 0.698, which is still significantly better than the other baselines. Our approach still detects unseen anomalies 63% better than just training on the source domain and 47.2% better than MixMatch as the best baseline on this dataset.

In addition, using the model selection approach we described in Section IV-B, as shown in Figure 7, we could further increase the detection rate to 0.737 based on the average F1 score which is 66.9% better than just training on the source domain and 51.1% better than the best baseline. More details on how PPPL detects anomalies in this scenario for each of the individual tasks can be found in Table V in the appendix.

G. Generalization of PPPL on Other Datasets

In this section, to evaluate how well PPPL generalizes, we report its performance on three other datasets that we built based on public datasets created for the evaluation of NIDS. In these datasets, the features for each record are extracted using very different feature extractors compared to the one we used in our earlier evaluations on the CICIDS2017 dataset.

1) Datasets: For evaluation of the generalization of PPPL, we created 3 different domain adaptation tasks as follows:

Case 1: For this case, we used the CIC-ToN-IoT dataset [37] to build our source domain and target domain. The source domain consists of 50,000 benign records which are randomly sampled from all the benign records and 5,000 malicious records which are randomly sampled from the *password* class. The target domain consists of another 50,000 benign records which are randomly sampled from the remaining benign records and 5,000 malicious records which are randomly sampled from the *ransomware* class. So, the goal, in this case, is to see how well a model trained on password attacks can detect

TABLE II: Comparison of PPPL with other baselines on the CIC-ToN-IoT and NF-UNSW-NB15 datasets. The numbers reported in this table are F1 scores.

Method	Case 1	Case 2	Case 3
Only-Src	0.052	0.000	0.488
MixMatch	0.000	0.000	0.4382
PPPL S1	0.689	0.957	0.682
PPPL S2	0.688	0.957	0.687
PPPL S3 Ensamble	0.596	0.458	0.681
PPPL S3 Model Selection	0.594	0.458	0.682

ransomware attacks. Each data point in the original CIC-ToN-IoT dataset had 83 features. Six of these features (Flow ID, Src IP, Src Port, Dst IP, Dst Port, and Timestamp) do not capture the intrinsic characteristics of the traffic. So we removed them from our dataset and used the remaining 77 features for training the models.

Case 2: For this case, we also used the CIC-ToN-IoT dataset. In this case, the source domain consists of 50,000 benign records which are randomly sampled from all the benign records, and 1,000 malicious records which are randomly sampled from the *injection* class. The target domain consists of another 50,000 benign records which are randomly sampled from the remaining benign records and 5,000 malicious records which are randomly sampled from the *backdoor* class. So, the goal, in this case, is to see how well a model trained on injection attacks can detect backdoor attacks.

Case 3: For this case, we used the NF-UNSW-NB15 dataset [38]. The source domain consists of 50,000 benign records which are randomly sampled from all the benign records and all the records from the *Shellcode* and *Analysis* attacks. The target domain consists of another 50,000 benign records which are randomly sampled from the remaining benign records and all the records from the *Worms*, *Backdoor*, and *DoS* attacks. So, the goal, in this case, is to see how well a model trained on shellcode and analysis attacks can detect worms, backdoor, and DoS attacks. Each data point in the original CIC-ToN-IoT dataset had 12 features. Four of these features (IPV4_SRC_ADDR, L4_SRC_PORT, IPV4_DST_ADDR, L4_DST_PORT) do not capture the intrinsic characteristics of the traffic. So we removed them from our dataset and used the remaining 8 features for training the models.

2) Implementation Details: For these three datasets, we used the same implementation details that we described earlier. The only difference is that, for these three cases, we did not downsample the benign inputs in each epoch of PPPL.

3) Results: Results of the evaluation of PPPL on these three datasets can be found in Table II. In this table PPPL S1 is for evaluation of our method in scenario 1, PPPL S2 is for evaluation of our method in scenario 2 when the error rate is set to be 70% and PPPL S3 is for evaluation in scenario 3. As can be seen in this table, even though these datasets have been created with very different feature extractors, PPPL can still detect unseen anomalies in the target domains and it significantly outperforms the baselines in all of these cases.

VI. Discussion

In this paper, for the first time, we proposed to formulate the problem of detecting unseen network attacks as a domain adaptation problem. As we showed, other domain adaptation methods provide very limited improvement when used for detecting unseen anomalies in network traffic. Therefore, we introduced PPPL as a new domain adaptation method that can work well on top of features extracted from network traffic. But, PPPL was not designed to only work for network traffic. It should also provide some improvement when used for other scenarios where we have datasets with similar characteristics (e.g., structured feature vectors, imbalanced dataset).

To use PPPL for detecting unseen anomalies, one first needs to collect a labeled dataset, which includes the normal behavior of the network in addition to some known attacks. Then a model can be trained using this dataset, and the PPPL method could be used to help us detect different types of network attacks in another un-labeled dataset that we are interested in.

As we have shown in our evaluation, while we can significantly improve the detection rate of unseen network attacks, we cannot completely distinguish between benign anomalies and malicious anomalies. However, leveraging a domain adaptation method such as PPPL in building an NIDS has the potential to make the system better distinguish between these two types of anomalies. In other words, our approach enables the model to become adapted to types of benign anomalies that happen gradually and eventually become the new normal over time. Without using a domain adaptation method, these benign anomalies would be flagged as malicious. Our approach has its own limitations and cannot distinguish between benign anomalies that are very different from current normal traffic and just happen for a short period of time. Also, to train a model using PPPL two existing datasets are required. Therefore, it cannot be trained on real-time network traffic. We leave working on these issues for future work.

VII. Related Work

Network intrusion detection systems (NIDS) focus on observing the network flow between computers to detect malicious anomalies between different computers in the network. NIDS are often categorized into three groups: signature-based, anomaly-based, and other ML-based techniques. We first give a short overview of signature-based and anomaly-based techniques, and then provide a thorough review of other ML-based NIDS, which is the main focus of this paper. The major difference between our method and other methods we discuss in this section is that our method incorporates both labeled data and unlabeled data in the training phase to detect unseen anomalies, whereas other methods used for detecting anomalies in the network traffic are either trained in an unsupervised fashion (with unlabeled data) or with a vanilla supervised method (i.e., only with labeled data). Table III shows methods used in each related work. We refer interested readers to the survey papers on NIDS such as Molina-Coronado et al. [39].

Signature-based Techniques. This technique uses known attacks and detects malicious flows by matching the observed traffic to a known attack [40]. Since these techniques rely on known patterns, they are known to be ineffective in identifying zero-day attacks [41], [42]. Therefore, other approaches are often used to generalize and detect new classes of attacks.

TABLE III: Methods used in the related work

System	Methods
Anomaly-based NIDS	1. Grey Wolf Optimizer + Convolutional Neural Network [45] 2. Deep Reinforcement Learning [47] 3. Adversarial Autoencoders [49] 4. Long Short-Term Memory + Convolutional Neural Network [50] 5. Graph Neural Network [52] 6. Graph Neural Network + Generative Adversarial Networks [53]
ML-based NIDS	1. Support Vector Machines [54] 2. Decision Trees [55], [56] 3. K-Nearest Neighbors [57] 4. Bayesian optimization [58] 5. Radial Basis Function (RBF) [59] 6. Gradient-based Feature Engineering [60] 7. Relevance Vector Machine [61] 8. Multi-Layer Perceptron [62] 9. Gated Recurrent Unit [62] 10. Self-Organizing Maps [62]

Anomaly-based Techniques. The techniques based on anomalies define normal behaviors and use them to detect malicious flows that depart from the norm [43]. Statistical-based, knowledge-based, and machine learning-based methods are pervasively adapted to infer the normal model of system behaviors in an offline mode and use them to detect anomalies online [44]. Garg et al. [45] proposed a technique that combines grey wolf optimizer [46] for feature selections with convolutional neural networks (CNN) for training with minimum feature engineering. Their experiments show that the hybrid model outperformed the standard CNN as well as traditional techniques such as support vector machine (SVM). DeepGuard [47] proposed a deep reinforcement learning (Deep-RL) technique to monitor network traffics for anomaly detection. At run-time, they use a Deep-RL controller as a policy network to match normal traffic flow, maximizing both performance and security. Adran [48] uses adversarial autoencoders [49] to learn the normal behavior of network flows in the latent representation with arbitrary distributions. CANnolo [50] uses Long Short-Term Memory (LSTM)-autoencoders to learn the normal behavior of Controller Area Networks (CANs). At runtime, it computes the difference between the reconstructed and the real sequence to detect anomalies. Abdallah et al. [51] proposed a hybrid model of convolutional neural network (CNN) with a long short-term memory network (LSTM) to detect anomalies in software-defined networks (SDNs). They showed that such combinations improve the precision of detecting new intrusions to the SDNs, which are not being seen during the training. Rather than inferring normal behaviors and detecting anomalies by measuring the deviation of traffics from the norm, we focus on training ML models that discriminate between normal and anomaly traces that go beyond the training distributions and detect unknown patterns of attacks.

Anomaly detection techniques have been also proposed beyond network traffic analysis [52], [53]. Recently, Zhou et al. [52] proposed a graph-based learning method to detect unseen anomaly nodes on networks. Their main observation is that the unseen anomaly is often closely mixed with normal behaviors, whereas the seen anomaly is often well-discriminated from the normal ones. Therefore, they require learning a normal model of networks that has complex representations to be separated from unseen anomalies, while it pushes away a seen anomaly with a simpler representation. First, they divide datasets into five groups as follows: the normal training set, seen anomaly training set, the normal test set, seen anomaly test set, and the unseen anomaly test set. Then, they adapt graph neural networks (GNNs) to represent graph nodes in low-dimensional representation in latent space. Then, they used Gaussian Mixture Model (GMM) algorithms to further cluster nodes into fine-grained patterns to effectively discriminate unseen abnormal patterns from other classes. Finally, they used multi-hypersphere graph learning to infer normal behaviors where the objectives are to discriminate normal nodes from unseen anomaly nodes while pushing seen anomaly nodes far away. Ding et al. [53] tackle the problem of finding anomalies in attributed networks that generalize to unseen nodes. They focus on modifying a standard graph neural network (GNN) to represent anomaly nodes and use generative adversarial networks (GANs) to generate new data including anomaly ones that deviate from the training distribution. Our approach is crucially different in that we specifically focus on network traffic (i.e., dynamic behavior of latent graph models) rather than abnormal nodes in the arbitrary network graphs (i.e., static behavior of underlying graphs).

ML-based Techniques. As illustrated by Tsai et al., [63], traditional machine learning techniques such as support vector machine (SVM) [54], decision trees [55], [56], k-nearest neighbor (KNN) [57], Bayesian optimization [58], and radial basis function (RBF) [59] have been significantly used to detect network attacks. Injadat et al. [58] used novel training sample size reductions, feature engineering, and hyperparameter tuning to improve the performance of network intrusion detection systems. Upadhyay et al. [60] used gradients to identify important features and automate the feature engineering step in intrusion detection systems. In addition to being unable to detect anomalies from unknown distributions, these techniques still require feature engineering to infer an accurate NIDS classifier. Closer to our work, Wu et al. [61] proposed an adaptive intrusion system that used ensemble incremental learning to detect attacks including those with possible distribution shifts. In particular, they used an ensemble of Relevance Vector Machine (RVM) [64] to represent different data distributions and adapt to the new data distribution by taking the weighted majority vote. However, their approach is limited to simple linear functions and cannot generalize to complex decision boundaries. In contrast, our approach used deep neural networks with domain adaption that can infer functions with arbitrarily complex shapes and do not require feature engineering. Di Mauro et al. [62] performed experiments on the effectiveness of various state-of-the-art deep learning in detecting network intrusions. The experiments include multi-layer perceptron (MLP), convolutional neural networks (CNNs), recurrent neural networks (RNNs), long short-term memory (LSTM), gated recurrent unit (GRU) [65], WiSARD [66], learning vector quantization (LVQ) [67], and self-organizing maps (SOM) [68]. In doing their experiments, they consider coarse-grained features (e.g., source and destination Port), time-based features (e.g., inter-arrival times between two flows), flow-based features (e.g., length of a flow), packet-based features (e.g., number of packets in a flow), and flag-based features (e.g., number of packets with active TCP/IP flags). Overall, they found that the MLP-based architectures, CNN, RNN, LSTM, and GRU show a high accuracy (an average of 99%) in the single-class dataset such as DDoS attack. With different classes of attacks in the dataset (DDoS, Portscan, and Adware), they found that simpler DNN architectures such as MLP show the highest overall accuracy. However, Di Mauro et al. [62] did not consider the performance of deep learning architectures when inferred for unknown attacks with different distributions.

VIII. Conclusion

In this paper, we proposed Proportional Progressive Pseudo-Labeling (PPPL), a simple and novel method for domain adaptation that can be used to build a more accurate NIDS capable of detecting unseen anomalies in network traffic. We showed how the knowledge about target domain class proportions can be leveraged to better detect unseen anomalies, and we also showed how PPPL can be modified to work in more realistic scenarios in which no prior knowledge about the ratio of malicious traffic exists. PPPL aims to progressively reduce the error on the target domain by assigning pseudo-labels to the target domain samples and training the model with them while excluding samples with more likely wrong pseudo-labels from the training set and also postponing training on such samples. Experiments on a network traffic dataset consisting of multiple different attacks confirm the superiority of our approach compared to other strong baselines in detecting unseen anomalies with up to 58.4% improvement in detection rate based on the average F1 score.

IX. Acknowledgements

This work was supported in part by NSF and VMware as part of the NSF SDI-CSCS program (grant number 1700527).

Appendix

Details of PPPL's performance for each individual task in the second and third scenarios can be found below:

TABLE IV: Results of PPPL on the CICIDS2017 dataset in scenario 2 where the ratio of malicious traffic in the target domain is known by different levels of error. The numbers reported in this table are F1 scores.

Method	Error	A → B	A → C	B → A	B → C	C → A	C → B	Avg
PPPL	No Error	0.967	0.634	0.821	0.661	0.818	0.956	0.810
	10%	0.968	0.616	0.818	0.669	0.777	0.961	0.801
	20%	0.966	0.632	0.824	0.667	0.792	0.961	0.807
	30%	0.967	0.596	0.826	0.671	0.739	0.960	0.793
	40%	0.969	0.587	0.830	0.666	0.712	0.962	0.787
	50%	0.970	0.586	0.834	0.665	0.672	0.960	0.781
	60%	0.970	0.580	0.830	0.666	0.665	0.961	0.779
	70%	0.966	0.588	0.824	0.671	0.683	0.963	0.782
Only-Src	-	0.055	0.215	0.028	0.087	0.010	0.016	0.068

TABLE V: Results of PPPL on the CICIDS2017 dataset in scenario 3 where there exists no prior knowledge about the ratio of malicious traffic in the target domain.

Task		A → B	A → C	B → A	B → C	C → A	C → B	Avg
Ensemble of models								
F1 score		0.888	0.612	0.569	0.499	0.676	0.946	0.698
Model selection								
Model 1 (w=0.1)	F1 score	0.622	0.611	0.278	0.328	0.773	0.918	0.588
	Generalization score	2.843	5.003	2.506	4.558	14.228	9.118	-
Model 2 (w=0.7)	F1 score	0.969	0.611	0.785	0.674	0.300	0.953	0.715
	Generalization score	3.532	4.521	3.196	4.110	11.503	9.876	-
Selected model	Number	2	1	2	1	1	2	-
	F1 score	0.969	0.611	0.785	0.328	0.773	0.953	0.737

REFERENCES

[1] Accenture, "Ninth annual cost of cybercrime study," https://www.accenture.com/us-en/insights/security/cost-cybercrime-study, 2019.

[2] S. Morgan, "Global cybercrime damages predicted to reach $6 trillion annually by 2021," https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021, 2020.

[3] ICS-CERT, "Cyber-attack against Ukrainian critical infrastructure," www.ics-cert.us-cert.gov/alerts/IR-ALERT-H-16-056-01, 2016.

[4] Markets and Markets, "Intrusion detection and prevention systems market," https://www.marketsandmarkets.com/Market-Reports/intrusion-detection-prevention-system-market-199381457.html, 2020.

[5] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 3320–3328.

[6] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset Shift in Machine Learning*. The MIT Press, 2009.

[7] G. Kang, L. Jiang, Y. Yang, and A. G. Hauptmann, "Contrastive adaptation network for unsupervised domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4893–4902.

[8] M. Long, Z. CAO, J. Wang, and M. I. Jordan, "Conditional adversarial domain adaptation," in *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, pp. 1640–1650. [Online]. Available: http://papers.nips.cc/paper/7436-conditional-adversarial-domain-adaptation.pdf

[9] G. French, M. Mackiewicz, and M. Fisher, "Self-ensembling for visual domain adaptation," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rkpoTaxA-

[10] H. Liu, M. Long, J. Wang, and M. Jordan, "Transferable adversarial training: A general approach to adapting deep classifiers," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 4013–4022. [Online]. Available: http://proceedings.mlr.press/v97/liu19b.html

[11] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *J. Mach. Learn. Res.*, vol. 17, no. 1, p. 2096–2030, Jan. 2016.

[12] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, "CyCADA: Cycle-consistent adversarial domain adaptation," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 1989–1998. [Online]. Available: http://proceedings.mlr.press/v80/hoffman18a.html

[13] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Deep transfer learning with joint adaptation networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 2208–2217.

[14] X. Xu, X. Zhou, R. Venkatesan, G. Swaminathan, and O. Majumder, "d-sne: Domain adaptation using stochastic neighborhood embedding," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[15] V. K. Kurmi, S. Kumar, and V. P. Namboodiri, "Attending to discriminative certainty for domain adaptation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[16] S. Sankaranarayanan, Y. Balaji, C. D. Castillo, and R. Chellappa, "Generate to adapt: Aligning domains using generative adversarial networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[17] Z. Pei, Z. Cao, M. Long, and J. Wang, "Multi-adversarial domain adaptation," in *AAAI Conference on Artificial Intelligence*, 2018.

[18] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 1180–1189.

[19] M. J. Hashemi and E. Keller, "General domain adaptation through proportional progressive pseudo labeling," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 155–162.

[20] I. Sharafaldin, A. H. Lashkari, , and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *4th International Conference on Information Systems Security and Privacy (ICISSP)*, 2018.

[21] A. Tarvainen and H. Valpola, "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep

learning results," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1195–1204.

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 2672–2680.

[23] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," in *Advances in Neural Information Processing Systems*, L. Saul, Y. Weiss, and L. Bottou, Eds., vol. 17. MIT Press, 2005. [Online]. Available: https://proceedings.neurips.cc/paper/2004/file/96f2b50b5d3613adf9c27049b2a888c7-Paper.pdf

[24] D.-H. Lee, "Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks," *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)*, 07 2013.

[25] S. Laine and T. Aila, "Temporal ensembling for semi-supervised learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=BJ6oOfqge

[26] M. Sajjadi, M. Javanmardi, and T. Tasdizen, "Regularization with stochastic transformations and perturbations for deep semi-supervised learning," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 1171–1179.

[27] V. Verma, A. Lamb, J. Kannala, Y. Bengio, and D. Lopez-Paz, "Interpolation consistency training for semi-supervised learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 3635–3641. [Online]. Available: https://doi.org/10.24963/ijcai.2019/504

[28] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. A. Raffel, "Mixmatch: A holistic approach to semi-supervised learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/1cd138d0499a68f4bb72bee04bbec2d7-Paper.pdf

[29] M. J. Hashemi, G. Cusack, and E. Keller, "Towards evaluation of nidss in adversarial setting," in *Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks*, ser. Big-DAMA '19, 2019, p. 14–21.

[30] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 213–226.

[31] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 97–105.

[32] M. Long, H. Zhu, J. Wang, and M. I. Jordan, "Unsupervised domain adaptation with residual transfer networks," ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 136–144.

[33] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[34] T. Miyato, S.-I. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: A regularization method for supervised and semi-supervised learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 1979–1993, 2019.

[35] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=r1Ddp1-Rb

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[37] M. Sarhan, S. Layeghy, and M. Portmann, "An explainable machine learning-based network intrusion detection system for enabling generalisability in securing iot networks," *ArXiv*, vol. abs/2104.07183, 2021.

[38] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "Netflow datasets for machine learning-based network intrusion detection systems," in *Big Data Technologies and Applications*, Z. Deze, H. Huang, R. Hou, S. Rho, and N. Chilamkurti, Eds. Cham: Springer International Publishing, 2021, pp. 117–135.

[39] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2451–2479, 2020.

[40] K. Scarfone, P. Mell *et al.*, "Guide to intrusion detection and prevention systems (idps)," *NIST special publication*, vol. 800, no. 2007, p. 94, 2007.

[41] J. Navarro, A. Deruyver, and P. Parrend, "A systematic survey on multi-step attack detection," *Computers & Security*, vol. 76, pp. 214–249, 2018.

[42] D. Chou and M. Jiang, "A survey on data-driven network intrusion detection," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–36, 2021.

[43] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[44] D. Chou and M. Jiang, "Data-driven network intrusion detection: A taxonomy of challenges and methods," *arXiv preprint arXiv:2009.07352*, 2020.

[45] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A hybrid deep learning-based model for anomaly detection in cloud datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 924–935, 2019.

[46] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0965997813001853

[47] T. V. Phan, T. G. Nguyen, N.-N. Dao, T. T. Huong, N. H. Thanh, and T. Bauschert, "Deepguard: Efficient anomaly detection in sdn with fine-grained traffic flow monitoring," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1349–1362, 2020.

[48] L. Nie, L. Zhao, and K. Li, "Robust anomaly detection using reconstructive adversarial network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1899–1912, 2021.

[49] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[50] S. Longari, D. H. Nova Valcarcel, M. Zago, M. Carminati, and S. Zanero, "Cannolo: An anomaly detection system based on lstm autoencoders for controller area network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1913–1924, 2021.

[51] M. Abdallah, N. An Le Khac, H. Jahromi, and A. Delia Jurcut, "A hybrid cnn-lstm based approach for anomaly detection systems in sdns," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ser. ARES 21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3465481.3469190

[52] S. Zhou, X. Huang, N. Liu, Q. Tan, and F.-L. Chung, *Unseen Anomaly Detection on Networks via Multi-Hypersphere Learning*, pp. 262–270. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611977172.30

[53] K. Ding, J. Li, N. Agarwal, and H. Liu, "Inductive anomaly detection on attributed networks," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, ser. IJCAI'20, 2021.

[54] F. Kuang, W. Xu, and S. Zhang, "A novel hybrid kpca and svm with ga model for intrusion detection," *Applied Soft Computing*, vol. 18, pp. 178–184, 2014.

[55] A. S. Eesa, Z. Orman, and A. M. A. Brifcani, "A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems," *Expert systems with applications*, vol. 42, no. 5, pp. 2670–2679, 2015.

[56] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based intelligent intrusion detection system in internet of vehicles," in *2019 IEEE global communications conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[57] W. Li, P. Yi, Y. Wu, L. Pan, and J. Li, "A new intrusion detection system based on knn classification algorithm in wireless sensor network," *Journal of Electrical and Computer Engineering*, vol. 2014, 2014.

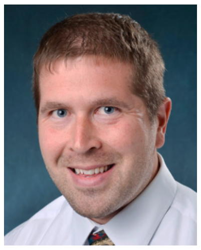
[58] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Multi-stage optimized machine learning framework for network intrusion detection," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1803–1816, 2020.

[59] R. Mills, A. K. Marnerides, M. Broadbent, and N. Race, "Practical intrusion detection of emerging threats," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 582–600, 2022.

[60] D. Upadhyay, J. Manero, M. Zaman, and S. Sampalli, "Gradient boosting feature selection with machine learning classifiers for intrusion detection on power grids," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 1104–1116, 2021.

[61] Z. Wu, P. Gao, L. Cui, and J. Chen, "An incremental learning method based on dynamic ensemble rvm for intrusion detection," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 671–685, 2022.
[62] M. D. Mauro, G. Galatro, and A. Liotta, "Experimental review of neural-based approaches for network intrusion management," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2480–2495, 2020.
[63] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *expert systems with applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.
[64] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *Journal of machine learning research*, vol. 1, no. Jun, pp. 211–244, 2001.
[65] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://aclanthology.org/D14-1179
[66] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, no. 1, p. 1–127, jan 2009. [Online]. Available: https://doi.org/10.1561/2200000006
[67] T. Kohonen, "Learning vector quantization," in *Self-organizing maps*. Springer, 1995, pp. 175–189.
[68] T. Kohonen, *Self-organizing maps*. Springer Science & Business Media, 2012, vol. 30.

Mohammad J. Hashemi received the B.S. degree from the Sharif University of Technology in 2014 and the M.S. and Ph.D. degrees from the University of Colorado Boulder in Computer Science in 2018 and 2021, respectively. He is currently a Postdoctoral Associate in the Electrical, Computer, and Energy Engineering department at the University of Colorado Boulder, and his research interests include adversarial machine learning, deep learning applications, and anomaly detection.



Eric Keller received the Ph.D. degree from Princeton University in 2011 and is currently an Associate Professor in the Electrical, Computer, and Energy Engineering Department at the University of Colorado Boulder. His research has been enabling and capitalizing on a more dynamic and programmable computing and network infrastructure, via such technologies as virtualization, software-defined networking, and the movement toward cloud-based services.

Saeid Tizpaz-Niari received the Ph.D. degree from University of Colorado Boulder in 2020 and is currently an Assistant Professor in the Computer Science Department at the University of Texas at El Paso. His research interests are at the intersection of Software Engineering, Machine Learning, and Cybersecurity where the goal is to automate the process of finding and explaining bugs, vulnerabilities, and fairness issues in large-scale software and machine learning systems.