Encrypted Network Traffic Classification using Self-supervised Learning

Md. Shamim Towhid
Department of Computer Science
University of Regina, Canada
mty754@uregina.ca

Nashid Shahriar
Department of Computer Science
University of Regina, Canada
nashid.shahriar@uregina.ca

2022 IEEE 8th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0694-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/NetSoft54395.2022.9844044

***Abstract*—Network traffic classification is used in many applications including network provisioning, malware detection, resource management, and so on. In modern networks, use of encrypted protocols is a norm rather than an exception. Existing network traffic classification techniques fall short in working with encrypted traffic. Although deep learning based techniques have been shown to perform well in the case of encrypted traffic classification, they require an abundance of labeled data to achieve high accuracy. However, labeled data is rarely available in sufficient volumes in real network settings as they require domain experts to annotate data with labels. Therefore, in this paper, we propose a self-supervised approach that can achieve high accuracy on encrypted network traffic classification with a few labeled data. The proposed method is evaluated on three publicly available datasets. The empirical result shows that our method not only achieves high accuracy on encrypted traffic but also has the ability to apply the acquired knowledge on a different dataset. In our experiments, our method outperforms the state-of-the-art baseline methods by ∼3% in terms of accuracy even with a much lower volume of labeled data.**

***Index Terms*—self-supervised learning, network traffic classification, encrypted traffic**

I. Introduction

Modern networks are becoming increasingly complex to support the heterogeneous needs of a variety of Internet applications. Automation is necessary to manage these complex networks. Network traffic classification is usually the first step in any automated network management system. It allows network operators to analyze traffic and identify different types of applications in the network which can be used to manage the overall performance of the network. Communication in the modern network is done by exchanging packets which are obtained by segmenting the data according to some protocol. As a result, there is a flow of these packets from one network device to another. Classification of this flow of network packets refers to this identification of applications of the traffic by observing the characteristics of the traffic flow. This identification of application can use multiple features in the network. For example, an Internet service provider can prioritize one particular type of flow over another, thus ensuring Quality of Service (QoS) for the prioritized network traffic. Some of the applications of network traffic classification are Quality of Service (QoS) provisioning, Service Function Chaining (SFC), malware detection, and anomaly detection [1].

Due to the various specific applications and the availability of different protocols, network traffic classification becomes a challenging task. Moreover, the development of encrypted protocols (e.g., HTTP/2 [2] and QUIC [3]) and wide acceptance of these protocols in network communication exacerbate the complexity of network traffic classification. In the past, traditional machine learning techniques showed promising performance in traffic classification. In [4], traditional machine learning (e.g., Naïve Bayes, Decision Trees) models are trained on a set of flow statistics selected by domain experts. Some more traditional machine learning approaches are mentioned in [5]. However, these approaches cannot work on encrypted network traffic because these techniques rely on hand-picked features, which cannot be easily extracted from encrypted traffic.

Recently, deep learning [6] is getting popular in the field of networking because of its success in other domains such as computer vision [7] and natural language processing (NLP) [8]. The main advantage of deep learning is that we do not need to design hand-picked features for it. These algorithms can learn the required features by themselves [6]. Most promising deep learning algorithms follow the supervised learning paradigm [9]. In supervised learning, we need to provide the actual label (a.k.a ground truth) for the sample data. Then, the algorithm can learn the important features from provided data using the label during training. However, creating this labeled data is time-consuming and errors may occur during labeling the data. One major drawback of supervised learning is that it requires lots of training data to train a model with good accuracy.

Another type of approach from the machine learning domain is recently getting popular, called self-supervised learning [10]. In self-supervised learning, we need a small amount of labeled data to get a model with good accuracy. These algorithms try to learn a representation of the data that is used for the downstream task later on. Usually, a self-supervised learning algorithm has two stages [10]–[12]. The first stage is called pre-training stage where the model learns to represent the data differently based on some property of the data rather than the actual label. The next stage is called fine-tuning stage where the weights of the model are fine-tuned based on a small amount of labeled data of the actual task. The idea is that if the model can learn a good representation of the data in the pre-training stage, a small amount of labeled data will be sufficient in the fine-tuning stage to achieve high accuracy. The overall accuracy of the model highly depends on the given task and the amount of unlabeled data in the pre-training stage. This type of algorithm is suitable for encrypted network traffic classification because it is easy to capture lots of traffic data from a network but the tedious part is to create the label for every captured flow of packets. In self-supervised learning, we need only a few labeled data along with a lot of unlabeled data. Therefore, self-supervised learning seems to be a great fit for encrypted network traffic classification [25].

Self-supervised learning is not a new concept. Researchers proposed many self-supervised learning methods in the past to solve various types of problems. Those algorithms were not as good as supervised learning. Recently, Ting Chen et al. propose a self-supervised learning method in [11] that outperforms its corresponding supervised learning for a computer vision task. Since then people are adapting self-supervised learning methods in various domains [12], [13]. In this paper, we are adopting a well-known self-supervised learning method proposed in [12] and enhancing it further to address encrypted network traffic classification problem. According to our knowledge, this is the first attempt to use a self-supervised learning method for encrypted network traffic classification. Our main contributions are-

- We propose a novel method to apply self-supervised learning for encrypted network traffic classification using protocol agnostic features. So, our proposed method can work with traffic using any protocol.
- We show that our proposed method outperforms several state-of-the-art methods in terms of accuracy by evaluating our method on three different datasets [1], [20] and [24].
- We also showcase the transferability of our proposed method according to [1]. Once the model is trained, it can be used on a different dataset. This transferability is crucial for network traffic classification because network conditions and corresponding traffic characteristics may change over time. In this type of dynamic scenario, training the network from scratch every time is not a feasible solution. By using this transferability, our model can easily adapt to the new network condition. Our proposed method shows a slight decrease in accuracy when used in this setup, whereas an alternative method proposed in [1] faces significant decline in accuracy.

In the next section, we provide our literature review. We will discuss the motivation behind our work in this section. In section 3, we will describe every component of our method in detail. The evaluation of our method on three different datasets is given in section 4. Finally, we conclude the paper in section 5 by providing a summary of our work and discussing some future research directions.

II. Related Work

As network traffic classification is a well known problem, there is a plethora of research work on this topic. Many researchers have tried to apply traditional machine learning methods to distinguish between protocols (e.g., DNS, SMTP and HTTP) in a network trace [4], [5]. The accuracy of these traditional approaches has declined because of their simplicity and inability to capture complex traffic patterns [14].

Encrypted traffic classification is more challenging because it operates on non-standard port numbers. Deep learning is suitable for these kind of challenging tasks because we can feed large fine-grained feature vectors such as raw traffic to the models. These models can learn complex patterns from the raw traffic. From the deep learning domain Multilayer Perceptron (MLP), Stacked Autoencoders, CNN and LSTM have been explored extensively in recent literature [15]–[19]. Iman Akbari *et al.* [20] proposes stacked LSTM and CNN based tripartite neural network architecture to classify encrypted network traffic. Their approach uses three types of features namely flow statistics, raw bytes, and time series. For the time series features, they use inter arrival time of a packet in a flow, packet length, and direction. Since it is a supervised approach, their method needs a lot of labeled data to achieve good accuracy. On the other hand, only time series based features *i.e.* inter arrival time, packet size, and direction are used in [1]. This is a semi-supervised approach which requires fewer labels compared to supervised methods. In this paper, we compare our result directly with the semi-supervised approach of [1] as we are using the same dataset. In [1], the authors first train the CNN model to predict the statistics of the whole flow of packets using a sampled flow. They use three different sampling techniques in their paper which are fixed sampling, random sampling, and incremental sampling. We adopt their sampling techniques in this paper and use that as a form of augmentation for our proposed self-supervised method. With the same number of labeled data, our proposed method outperforms this semi-supervised approach in terms of accuracy. Furthermore, according to [1], in a transfer learning setup where the model is pre-trained using a different dataset, the accuracy drops almost 15%. In a similar setup, the accuracy of our proposed method drops only 2%.

In [24], A. S. Iliyasu *et al.* proposes another semi-supervised approach for encrypted traffic classification. A deep convolutional generative adversarial network is used in their approach. However, the accuracy of their approach is lower than the other semi-supervised approach mentioned earlier in [1]. Another semi-supervised time series classification approach is proposed in [25]. The authors in [25] adopt the self-supervised approach proposed in [11] and apply that for time series classification. The problem with their approach is, it needs negative sample to be present in the same batch during the contrastive learning stage. A negative sample is a transformed version of another time series when the loss is calculated for a particular time series. Since we do not use actual label in the contrastive learning stage, we can not decide whether the selected negative sample is coming from a different time series or it is a different series from the same class. In [11], this problem is solved by assuming that if we select a large batch size for training, the selected negative sample will be from another class with a high probability. Our proposed method completely omits this requirement of negative samples by following [12]. In addition to that, no experiment is done in [25] to prove the transferability of the method.

Some other works [21] apply ensemble learning for network traffic classification. In [21], the authors propose an ensemble of CNN models. Fundamentally, the working principle of ensemble learning and self-supervised learning is not the same. Ensemble learning is a type of supervised learning which requires a lot of labeled data to achieve high accuracy. In [21], the ensemble method is trained using a lot of labeled data compared to our approach. Transformers [22] are becoming popular to deal with sequential data due to its huge success in NLP. A transformer based classification approach for time series is proposed in [23]. However, the ability to classify encrypted network traffic with a few labeled data of these types of methods is yet to be explored.

In this paper, we evaluate our proposed approach using three publicly available datasets used in [1], [20], and [24] and compare our approach with the semi-supervised approach of [1]. We also show transferability of our approach.

III. Methodology

Similar to most of the self-supervised techniques, our proposed method has two stages. In the first stage, we train a neural network with unlabeled data. We call it the pre-training stage. In the second stage, we fine tune the weights of the neural network with a few labeled data. This stage is called the fine-tuning stage. In this section, we describe both of these stages with the data preparation.

A. Data Preparation

Our method uses two types of features from a flow of network packets. One is inter arrival time (IAT) between two packets in a flow. The other is the direction of the packet. The direction can be either forward (from source to destination) or backward (from destination to source). Notice that these two features do not depend on the used protocols. Since we rely on publicly available datasets, we are restricted to use the available features in those datasets (described in section IVA). IAT, direction, and packet length are available in all the used datasets. In our experiments, the highest accuracy (98.01%) is achieved using IAT and direction as features, whereas IAT, direction, and packet length achieve slightly lower accuracy (96.65%) on the QUIC dataset, justifying our decision of selecting IAT and direction for training our models.

We need to prepare two sets of data. One unlabeled dataset ($\mathcal{D}_U$) for pre-training stage and another labeled dataset ($\mathcal{D}_L$) for supervised fine-tuning. In the pre-training stage, we apply augmentation on a flow to get a transformed version of it. Two flows from the same class are required in the pre-training stage because the goal here is to learn similar representation for these two flows coming from the same class. Since we do not have access to labels in the pre-training stage, we cannot select these two flows using labels. Therefore, augmentation is applied to create another instance of a flow from the same class. Augmentation on numerical data is harder as the label of the data may change after applying augmentation. Therefore, we divide a flow of packets into sub-flows as explained below. Then we use one sub-flow as an augmented version of another sub-flow. As a pre-processing step, we make sure that all the flows in our dataset have at least 300 packets in them. There are two benefits of creating sub-flows. The first one is, we can use one sub-flow as an augmented version of another. The other benefit is that training dataset size increases by creating sub-flows. According to [11], the pre-training stage training benefits from large number of unlabeled data.

To create sub-flows from a flow of packets, we follow the incremental sampling technique mentioned in [1]. This technique has three parameters α, β, and γ. It selects packets from a flow that are α packets away from each other. After selecting β number of packets, it multiplies α by γ. By following [1], we use $\alpha = 10$, $\beta = 22$ and $\gamma = 1.6$ for all our datasets. We also sample only 45 packets from the entire flow. That means the length of a sub-flow is 45.

In the pre-training stage, we will use one sub-flow as an augmented version of another sub-flow. These two sub-flows must come from the same flow. Otherwise, if a sub-flow comes from a flow of a different class, it is not possible to use the sub-flow as a form of augmentation. To keep track of the sub-flows, we include a file identification number, F_{id} to each of the sub-flow of a flow. In other words, F_{id} is unique for each of the flow in our dataset. The unlabeled dataset $\mathcal{D}_U$ consists of sub-flows and the file identification number corresponding to each of the sub-flows.

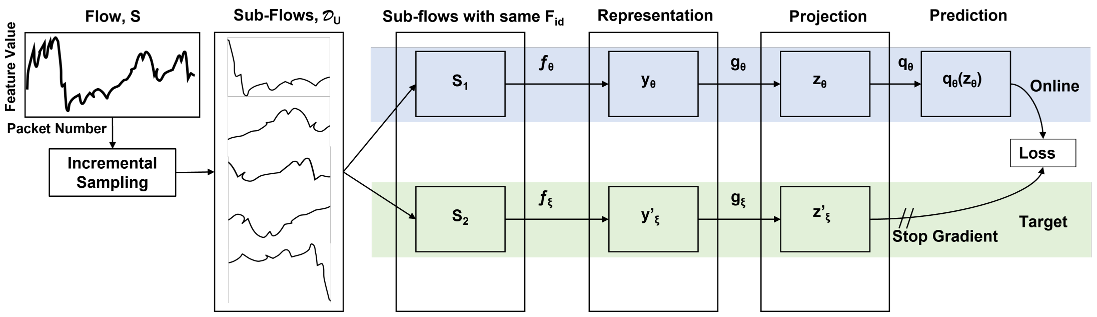
The labeled dataset, $\mathcal{D}_L$ is a smaller dataset compared to $\mathcal{D}_U$. For the Orange'20 [20] and QUIC [1] dataset, we have only 30 flows per class to put in $\mathcal{D}_L$. Note that these flows are not included in the set that is used to prepare $\mathcal{D}_U$. These 30 flows per class are randomly selected from the set of all flows. The length of each sub-flow in $\mathcal{D}_L$ must be the same as the length of a sub-flow in $\mathcal{D}_U$. Otherwise, we can not use the same trained model from pre-training stage. Therefore, we prepare sub-flows of $\mathcal{D}_L$ by following the same incremental sampling technique mentioned above. The only difference is, we include class labels for each of the sub-flows of $\mathcal{D}_L$. $\mathcal{D}_L$ is used in the fine-tuning stage as supervised dataset.

B. Pre-training Stage

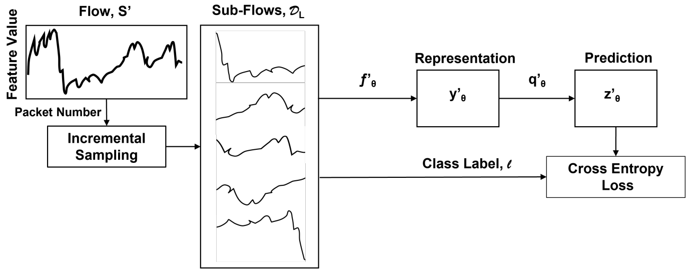
After preparing the unlabeled dataset $\mathcal{D}_U$, now it is time to start the training. Fig. 1a shows every step of the pre-training stage. By following [12], we use two neural networks in the pre-training stage. One network is called the online network and the other is named as target network. The purpose of the target network is to provide the regression target for the online network.

Both of the neural networks in the pre-training stage have one encoder f, and one projector g. The architecture of these two components is the same but they are initialized with different sets of weights. Here, θ represents the weights of the online network and ξ represents the weights of the target network. The online network has one extra layer for prediction. The dimension of the representation vector is very high. The purpose of the projectors g_θ and g_ξ is to convert this high dimensional vector to a low dimension.

In our work, we use ResNet [26] architecture as the encoder (f_θ and f_ξ). The goal in the pre-training stage of



(a) Pre-training Stage



(b) Fine-tuning Stage

Fig. 1: Overview of the proposed method

self-supervised learning is to separate different classes of data in the feature space by learning such representation of the data. ResNet is a popular choice of encoder in self-supervised learning due to it's powerful representational ability by increasing the depth of the model. ResNet has some variations such as ResNet18, ResNet34, ResNet50 and so on [26]. ResNet34 is selected for our training because it provides a good trade-off between accuracy and training time compared to the other variations. All the 2D convolution layer and batch normalization layer of ResNet34 is replaced by 1D convolution layer and batch normalization layer respectively. For the projectors (g_θ and g_ξ), fully connected layers are used. In the online network, the prediction layer is also a fully connected layer.

In the pre-training stage, we take two sub-flows (S_1 and S_2) from $\mathcal{D}_U$ with the same F_{id}. One sub-flow S_1 goes to the online network and the other sub-flow S_2 goes to the target network. From the encoders f_θ and f_ξ, we obtain the representations of the two sub-flows. Then these two representations go to the corresponding projectors (g_θ and g_ξ). From there, we obtain a low dimensional vector representation. Finally, the predictor q_θ of the online network predicts the representation of the target network which is z'_ξ. To be able to predict the representation of the target network, the online network needs to learn a representation which is close to the target network. Note that the input to these two networks are sub-flows coming from the same actual flow. Therefore, training in this way helps the encoder to learn a similar representation for the sub-flows coming from the same flow. Furthermore, sub-flows from different flows are represented differently in the feature space. Since the model is already learning to represent sub-flows from the same flow close to each other in the feature space, a few labeled data will be enough to achieve high accuracy in the fine-tuning stage. During the training, the weights of the online network are updated in every step of the training. But the weights of the target network is an exponential moving average of the weights of the online network. A target decay rate $\tau \in [0, 1]$ is selected to calculate this exponential moving average according to the following equation.

$$\xi \leftarrow \tau\xi + (1-\tau)\theta \tag{1}$$

We use the same symmetric loss function from [12]. In this loss function, one mean squared loss $\mathcal{L}_{\theta,\xi}$ is defined between the normalized predictions of the online network and the normalized projections of the target network. The mean squared loss is given in Eq. 2.

$$\mathcal{L}_{\theta,\xi} \triangleq \left(\frac{q_\theta(z_\theta)}{\| q_\theta(z_\theta) \|_2} - \frac{z'_\xi}{\| z'_\xi \|_2}\right)^2 \tag{2}$$

In Eq. 2, the loss is defined as the squared difference between l_2 normalization of predictions of the online network ($q_\theta(z_\theta)$) and projections of the target network (z'_ξ). Note that the loss in Eq. 2 is calculated by feeding S_1 to the online network and S_2 to the target network. To make the loss symmetric, another version of it is calculated by feeding S_1 to the target network and S_2 to the online network. This loss is denoted as $\widehat{\mathcal{L}_{\theta,\xi}}$. So, at each iteration a stochastic optimization step is performed to minimize the total loss $\mathcal{L}^{\mathcal{TOTAL}}_{\theta,\xi} = \mathcal{L}_{\theta,\xi} + \widehat{\mathcal{L}_{\theta,\xi}}$ with respect to θ only. Finally, we update θ according to the following equation.

$$\theta \leftarrow optimizer(\theta, \nabla_\theta \mathcal{L}^{\mathcal{TOTAL}}_{\theta,\xi}, \eta) \tag{3}$$

Here, η is the learning rate. In all our experiments, we use $\eta = 0.0001$ with Adam [29] optimizer. A weight decay rate of $1e^{-6}$ is used with the optimizer. In pre-training stage, we train the network for 50 epochs with a batch size 128. The target decay $\tau = 0.99$ is used. From the encoder, we obtain a representation vector of size 4096. In the projection layer, we reduce this representation vector to a 256 dimensional vector.

C. Fine-tuning Stage

From the pre-training stage, we obtain the trained encoder for the online network (f_θ). In the fine-tuning stage, we freeze all the layers of the encoder. By freezing, we mean there is no update to the weights of the encoder layers in this stage. The frozen layers are represented by f'_θ in Fig. 1b. We use the small supervised dataset, $\mathcal{D}_L$ in this stage. A new fully connected layer q'_θ is added as a prediction layer on top of the encoder. The number of neurons in this fully connected layer is equal to the number of class in our dataset $\mathcal{D}_L$. In this stage, we use the cross entropy loss (Eq. 4) to update the weights of the prediction layer q'_θ.

$$\mathcal{L}(l, z'_\theta) \triangleq -\sum_{\forall x} l(x)\log(z'_\theta(x)) \tag{4}$$

In Eq. 4, x represents one single observation from the distribution. l represents the true distribution and z'_θ represents the predicted distribution from the prediction layer.

In this stage, we train the network for 25 epochs with a batch size of 128. We use the Adam optimizer with a learning rate $1e^{-4}$ and weight decay $1e^{-6}$. Our evaluation metrics and the result of our experiments are presented in the next section.

IV. Evaluation

In this section, we evaluate our proposed method. At first, we discuss the used datasets in our experiments. The number of flows used in the pre-training stage and fine-tuning stage are shown in this section. Next, we discuss the result of our method on the selected datasets. We also explore how the accuracy of our method changes with different length of sub-flows and number of labeled data in the fine-tuning stage. Finally, we conclude the section by showing the transferability of our method.

Like most of the classification models in literature, we use accuracy, F1 score, recall and precision as our evaluation metrics. To explain these evaluation metrics, we will use the following terms.

- True Positive (TP): The number of times the model predicted positive for class n and it is true.
- True Negative (TN): The number of times the model predicted negative for class n and it is true.
- False Positive (FP): The number of times the model predicted positive for class n and it is false.
- False Negative (FN): The number of times the model predicted negative for class n and it is false.

For a binary classification problem like Orange'20 where we have only two classes, it is easy to understand the above definitions. For a multi-class problem like QUIC where we have more than two classes, we calculate each of the above term per class. Then we calculate an evaluation matrix per class using the following formulas. Finally, we take the average of all the classes to get the final value of that particular evaluation matrix. This is called the "macro" average in the literature [21].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$F1 = 2\frac{Precision}{Precision + Recall}$$

We use Scikit-learn [30] for the evaluation metrics and Pytorch Lightning [31] for the implementation of our method. All the experiments are performed on an 11^{th} Gen i7 machine that has one GeForce RTX 3090 GPU with 24GB memory and 32GB main memory. As mentioned in the "Methodology" (see IIIC) section, we train for 25 epochs in the fine-tuning stage. After each epoch, we calculate the evaluation metrics on the test set. The best values are recorded as the result for that training. We run the fine-tuning stage five times and take the average of the evaluation metrics as the final result.

A. Dataset

Encrypted protocols such as QUIC are only recently being adopted in practice, hence it is hard to find suitable datasets for encrypted network traffic classification. To evaluate our approach, we use QUIC, Orange'20 and ISCX VPN-NonVPN dataset from [1], [20] and [27], respectively. QUIC and Orange'20 datasets are published by the corresponding authors of the papers. ISCX dataset is published by the Canadian institute of cyber security. We use the QUIC dataset as it is from [1] and pre-process the Orange'20 and ISCX dataset as described below. In Orange'20 dataset, the number of flows per application level class is highly imbalanced after the pre-processing step. For example, FacebookChat class has only two flows in the dataset. In self-supervised method, we need a large number of unlabeled data from each class in the pre-training stage [11]. Therefore, we merge similar classes into one class. Finally, as shown in Table I, we obtain two classes (namely, Social and Streaming) for this dataset. We obtain Social class by merging Facebook, Twitter, and Instagram classes, and Streaming class consists of Youtube, Netflix, Snapchat, and Facebook classes from Orange'20 dataset. Note that Orange'20 dataset distinguishes between Facebook browsing traffic and Facebook streaming traffic. In our experiments, Facebook browsing and Facebook streaming traffic classes of Orange'20 are merged into Social and Streaming classes, respectively.

ISCX dataset contains packets captured by using different applications such as Facebook, Netflix, Twitter, etc., which are encrypted using different encryption protocols. By following

[24], we use the raw format (pcap) of this dataset. To convert raw packets to flow of packets, we use pkt2flow tool [28]. We use a script to extract the required features (IAT and Direction) for our experiments. After getting the features from flow of packets, we follow the steps described in the "Data Preparation" section (see IIIA) of this paper to create sub-flows. After creating sub-flows, the number of data per class was extremely low. In [24], the same problem with ISCX dataset is addressed by the authors. Therefore, we follow the same strategy as [24] and combine similar services into one class. Finally, we obtain three classes for the ISCX dataset (namely, Chat, Streaming, and VOIP). Table I shows the number of flows and sub-flows for each of the datasets.

Dataset	Class Names	Number of flows	Number of Sub-flows, $\mathcal{D}_U$	Number of Sub-flows, $\mathcal{D}_L$
QUIC	Google Doc	1251	99831	2993
	Google Drive	1664	133663	3000
	Google Music	622	49936	1123
	Google Search	1945	156725	3000
	Youtube	1107	88019	2933
	Total	**6589**	**528174**	**13049**
Orange'20	Social	2097	127782	7677
	Streaming	1669	104209	3615
	Total	**3766**	**231991**	**11292**
ISCX	Chat	54	1205	221
	Streaming	12	213	79
	VOIP	10	176	54
	Total	**76**	**1594**	**354**

TABLE I: Number of flows and sub-flows in the datasets

Note that we use 30 flows per class to create supervised dataset $\mathcal{D}_L$ for QUIC and Orange'20 datasets. Among the selected 30 flows, 20 flows are used for training in fine-tuning stage and the rest of the 10 flows are used for testing the model. Due to insufficiency of data in the ISCX dataset, we use 10 flows from the streaming class, 50 flows from the Chat class, and 8 flows from VOIP class to create $\mathcal{D}_U$. The rest of the flows are used to prepare $\mathcal{D}_L$. From $\mathcal{D}_L$, we use 70% of the sub-flow for training the model in the fine-tuning stage and 30% of the sub-flow to test the model. As described in section IIIA, we create sub-flows from flows using incremental sampling. The number of created sub-flows in $\mathcal{D}_U$ and $\mathcal{D}_L$ are also shown in Table I. From the total number of sub-flows in each dataset, we can say that the selected three datasets represent three use cases. First, the QUIC dataset represent the best scenario where we have sufficient number of sub-flows. Second, the Orange'20 dataset has a moderate number of sub-flows. Finally, the ISCX dataset represents the worst scenario where we have a small amount of sub-flows available for the training.

B. Comparison with the baseline methods

For QUIC dataset, we can directly compare the result of our method with the result of semi-supervised technique proposed in [1]. As their source code is publicly available, we run their code on the QUIC dataset to get the result. The obtained result is reported in Table II as baseline for QUIC dataset. On the other hand, for Orange'20 dataset, we can not compare our result with [20] directly because the number of classes in our version of the dataset are not same as [20]. Similar to Orange'20, we can not compare our result on ISCX with [24] because after applying pkt2flow on our collected data, the number of classes are not the same as [24]. Therefore, we use the corresponding supervised trained model as a baseline for Orange'20 and ISCX dataset. To create the baseline for Orange'20 and ISCX, we train the ResNet34 model on $\mathcal{D}_L$ in a supervised manner. In Table II, these models are mentioned as "Supervised Baseline". The semi-supervised method from [1] is mentioned as "Semi-supervised" in Table II. In our first experiment, we apply our proposed method on the three individual datasets and compare the results with the baseline methods.

Method	Dataset	Accuracy (%)
Semi-supervised Baseline [1]	QUIC	95.11
Self-Supervised (Ours)	QUIC	**98.01**
Supervised Baseline	Orange'20	80.11
Self-supervised (Ours)	Orange'20	**82.34**
Supervised Baseline	ISCX	**86.73**
Self-supervised (Ours)	ISCX	86.56

TABLE II: Performance of different methods

As mentioned earlier, to record the result of our method, we run the fine-tuning stage five times. Then record the average accuracy in Table II. The standard deviations of these five runs are also calculated and shown as error bars in Fig. 2. Table II shows that our method outperforms semi-supervised approach proposed in [1] on QUIC dataset by 3%. It is worth to mention that the proposed semi-supervised technique in [1] uses IAT, direction, and length of the packet as features. We report their result using the same features whereas in our method we use only IAT and direction as features as these two features give us the best result. Using IAT, direction, and length as [1], we obtain accuracy of 96.65% from our method. Although the accuracy of Orange'20 is lower than QUIC dataset, our method outperforms the supervised baseline on Orange'20 dataset. The number of sub-flows used in the pre-training stage for Orange'20 dataset is smaller than QUIC dataset. This could be a potential reason for this low accuracy in the Orange'20 dataset. This statement is justified in the next experiment. The result on ISCX dataset is interesting. We see the supervised baseline performs slightly better than our method for this dataset. Since the number of flows in ISCX dataset is very low compared to the other two datasets, our method can not learn the representation well in the pre-training stage. Therefore, the performance is similar to supervised baseline. From this result on ISCX dataset, we can verify the statement that our approach needs a lot of unlabeled data in the pre-training stage.

Figure 2 shows the other evaluation metrics of our method for the three datasets with the calculated standard deviation of the five results. From the figure, the scores of all the evaluation metrics are close to each other for the same dataset. Therefore, we can conclude that our method is capable of classifying sub-flows of packets.

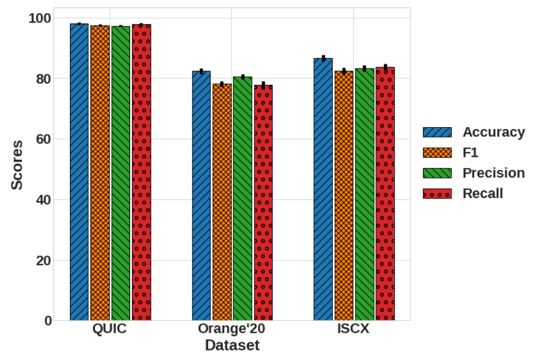


Fig. 2: Evaluation Metrics of our method

C. *Effect of varying the length of sub-flow*

Previously, we mentioned that the length of each sub-flow in our dataset is 45. In this experiment, we vary the length of the sub-flow. Decreasing the length of the sub-flow means we have less number of packet's information in a sub-flow. On the other hand, increasing the length of the sub-flow means we have more packet's information. Increasing the length should give us better accuracy as we have more packet's information in a single sub-flow. But it will increase the computation complexity. As a result, every epoch of the training will take longer time compared to the smaller length sub-flows. Table III shows the accuracy with different length of the sub-flows. The time per epoch in the fine-tuning stage is negligible compared to the pre-training stage as only one model is trained in the fine-tuning stage as opposed to two models being trained in parallel in the pre-training stage. In addition, the number of sub-flows used in the fine-tuning stage is much smaller compared to the pre-training stage. Since the accuracy of our method on ISCX dataset is lower than corresponding supervised model, we exclude ISCX from this experiment.

Dataset	Length of Sub-flow	ACC. (%)	Time per epoch (min)	
			Pre-training	Fine-tuning
QUIC	20	87.79	7.10	0.091
	45	98.01	15.16	0.157
	70	99.24	30.23	0.283
Orange'20	45	82.34	3.71	0.116
	70	83.20	8.12	0.241
	100	83.41	17.67	0.493

TABLE III: Varying the length of sub-flow

Table III shows, the accuracy increases with the increasing length of the sub-flow. But this comes at the cost of increased time taken for each epoch. The overall accuracy on Orange'20 is lower than the accuracy on QUIC dataset. Therefore, rather than decreasing the sub-flow length of Orange'20, we try to increase the length of sub-flows. Among the three cases (45, 70, and 100), we obtain the best result with the sub-flow length 100. We can not increase the length beyond 100 because the Orange'20 has some flows with length 300 and we need at least two sub-flows from incremental sampling for the pre-training stage. Although the accuracy increases a little with increasing length, still the accuracy of Orange'20 is lower than QUIC dataset. This behavior can be explained from the number of sub-flows in the pre-training stage. From Table I, we can see that the QUIC dataset has a larger amount of sub-flows than the Orange'20 dataset. As a result, the Orange'20 dataset has a lower amount of unlabeled data compared to the QUIC dataset in the pre-training stage. In the case of the Orange'20 dataset, the model cannot learn the representation well from the unlabeled data in the pre-training stage, thus yielding a lower accuracy. This result suggests that instead of increasing the length of the sub-flows, our method needs a lot of unlabeled data in the pre-training stage to achieve high accuracy. Table III shows that increasing the sub-flow length beyond 45 has a small gain in accuracy, whereas the increase in time per epoch is much more prominent. Therefore, we select 45 to be the best sub-flow length for our experiments.

Dataset	Length of Sub-flow	Classification time for one batch (s)
QUIC	20	0.009
Orange'20	45	0.016
Orange'20	70	0.022
Orange'20	100	0.028

TABLE IV: Classification time of the sub-flows

To be able to use the trained model in an actual network, it is important to know how much time it takes to classify sub-flows. In Table IV, we show the time (in seconds) to classify one batch of sub-flows. In our experiments, we set the batch size to 128 sub-flows. The classification time does not change based on the dataset. It depends on the length of the sub-flows. Therefore, Table IV shows classification time for all the different sub-flows length in our experiments.

D. *Effect of varying the size of the datasets*

In this experiment, we try to see the effect of changing the number of labeled data in the fine-tuning stage of our method. After the pre-training stage, we fine-tune the model using 5, 10, 15, and 20 flows per class in the supervised training set. Since there are 30 flows per class in the fine-tuning dataset, we use the rest of the flows for testing the model. For example, when we train with 5 flows per class in the fine-tuning stage then the model is evaluated on the rest of the 25 flows per class. Note that the number of flows mentioned here are divided into sub-flows using incremental sampling technique (described in section IIIA) before the fine-tuning stage. As a result, the actual volume of labeled data in the fine-tuning stage is different than the number of flows mentioned in Fig. 3. For example, when 5 flows per class are used, 2200 sub-flows are obtained for QUIC dataset and 2615 sub-flows are obtained for Orange'20 dataset. The model uses these sub-flows for training in the fine-tuning stage. Similarly, to evaluate the model during testing, sub-flows generated from chosen flows are used. We also try to see the effect of pre-training in this experiment. To do that, using the same dataset, we train our model without pre-training stage. This is similar to the supervised baseline models mentioned in section IVB. Fig. 3 shows the result of this experiment. From Fig. 3, it is clear that pre-training stage helps the model to achieve high accuracy. The model with pre-training stage outperforms the model without pre-training stage in all the cases for Orange'20 dataset. We see a similar behavior for the QUIC dataset. Only the last case, where we use 25 flows per class for training the model in the fine-tuning stage has the same accuracy (100%) with and without pre-training. From this result, we conclude that the test set is extremely small (only 5 flows per class are used for testing) to represent the whole dataset. Therefore, in this case, the model without pre-training can also achieve high accuracy. Otherwise, in all the cases, the model with pre-training is outperforming the model without pre-training by 3.74% on average.

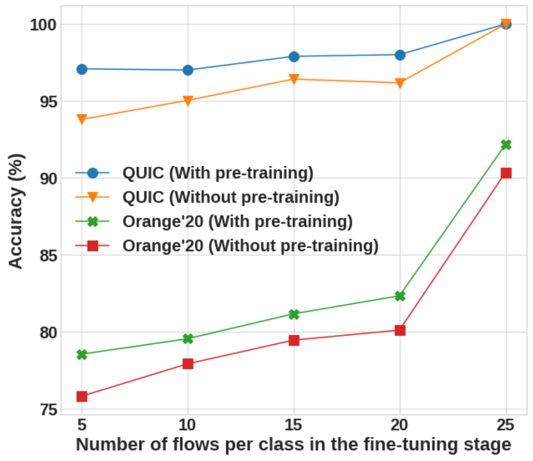


Fig. 3: Effect of varying the size of the fine-tuning dataset

In the previous experiment, the dataset size for pre-training was fixed while changing the dataset size for the fine-tuning stage. In the next experiment, we show the effect of pre-training by varying the dataset size of the pre-training stage while fixing the dataset size for the fine-tuning stage. Table V shows that the accuracy drops ∼5% when 10% of the total sub-flows (mentioned as $\mathcal{D}_U$ in Table I) are used in the pre-training stage compared to the case when 100% of the sub-flows are used in the pre-training stage. Therefore, we can conclude that the dataset size of the pre-training stage affects the overall accuracy of the model.

Dataset	Percentage of sub-flows ($\mathcal{D}_U$ in Table I) used for pre-training (%)	Accuracy (%)
QUIC	10	94.52
	30	94.81
	50	95.00
	100	98.01
Orange'20	10	77.16
	30	77.34
	50	79.02
	100	82.34

TABLE V: Effect of varying size of the pre-training dataset

E. Transferability of our method

In this experiment, we aim to show the transferability of our method. Transferability means whether the model can learn representation from a dataset and use that learned representation to classify network traffic from another dataset. Since the model learns representation of one unlabeled dataset ($\mathcal{D}_U$) in the pre-training stage, a different labeled dataset ($\mathcal{D}_L$) is used in the fine-tuning stage that utilizes the learned representation and labels from $\mathcal{D}_U$ and $\mathcal{D}_L$, respectively, to classify the network traffic. For example, the third column of the first row in Table VI shows the accuracy of the model when the QUIC dataset is used in the pre-training stage and Orange'20 is used in the fine-tuning stage. The fourth column of Table VI shows the accuracy when the same dataset (mentioned in the second column) is used for both pre-training and fine-tuning stages.

Pre-training Dataset	Fine-tuning Dataset	Accuracy(%) using different datasets	Accuracy(%) using same dataset
QUIC	Orange'20	81.48	82.34
Orange'20	QUIC	97.18	98.01
QUIC + ISCX + Orange'20	Orange'20	80.92	82.34
QUIC + ISCX + Orange'20	QUIC	97.83	98.01
QUIC + ISCX + Orange'20	ISCX	86.31	86.56

TABLE VI: Transferability of our method

Since ISCX dataset has a few number of flows, this dataset is excluded from the single pre-training experiment. For Orange'20 and QUIC dataset, the accuracy outperforms the baseline methods from Table II in both the cases whether we use single dataset for pre-training or combined dataset. This result suggests that our method can utilize the acquired knowledge from the pre-training stage even if the dataset is different. However, the accuracy on ISCX dataset drops when we use the combined dataset for pre-training. The accuracy is even less than self-supervised training mentioned in Table II. Since we have large number of flows in QUIC and Orange'20 datasets compared to ISCX, in the pre-training stage, the learned representations are dominated by Orange'20 and QUIC dataset. Therefore, the accuracy on ISCX dataset is less than what we get in Table II. Also, this is the reason why we exclude the cases of combining ISCX with Orange'20 or QUIC individually in this experiment. The result shown in Table VI suggests that if we include ISCX to any of the other datasets (QUIC or Orange'20) the model will learn most of the representations from that dataset rather than ISCX.

We now compare the transferability of our approach with that of [1]. When the proposed method of [1] is pre-trained on different dataset (Waikato [32]), the best accuracy of their method on QUIC dataset drops to 80.76% from 98.53% (reported results from [1]). On the other hand, as shown in Table VI, the accuracy of our proposed method drops from 98.01% to 97.18% as in the case of using Orange'20 in pre-training and QUIC in fine-tuning. On an average, the accuracy of our proposed method drops ∼2% when pre-trained on a different dataset and fine-tuned on different datasets. We also use the aggregate of the three datasets in the pre-training stage and each of the individual datasets in the fine-tuning stage (see last three rows of Table VI). In all the cases, the results are consistent. Therefore, we conclude that our proposed method outperforms the existing work [1] in terms of transferability.

V. Conclusion

In this paper, we propose a self-supervised method that uses a small amount of labeled data to classify encrypted network traffic. Labeling network traffic data is a time consuming and error-prone task. But collecting network traffic data is not that hard nowadays. In our method, we need a lot of unlabeled data and a few labeled data to be able to classify network traffic with high accuracy. Our method has two stages, namely pre-training and fine-tuning which are described in details. Then we evaluate the performance of our approach on three datasets. The results of all the experiments suggest that our method is able to achieve high accuracy with a few labeled data. At the same time, we need a lot of unlabeled data to get an overall high accuracy. Our proposed method is able to outperform the state-of-the-art baseline methods by $\sim$3% in terms of accuracy when there are enough data present in the pre-training stage. Furthermore, our method is able to achieve high accuracy when trained with unlabeled data from a different dataset than the labeled dataset, whereas the accuracy of the compared approaches decline significantly.

In our future research, the deployment process of the trained model in a real network will be investigated. We will address the problem of adding new class to an already trained model according to our method. It is often necessary to add a new class to a trained model without compromising the accuracy on existing classes. Explainability of the proposed method is needed to be explored as well. We will explore these capabilities of the proposed method in our future work.

Acknowledgement

We would like to thank our shepherd Dr. Sandra Scott-Hayward and anonymous reviewers of IEEE Netsoft 2022 for their valuable feedback.

References

[1] S. Rezaei and X. Liu, "How to Achieve High Classification Accuracy with Just a Few Labels: A Semi-supervised Approach Using Sampled Packets," arXiv:1812.09761 [cs], Dec. 2018.

[2] Mike Belshe, Roberto Peon, and Martin Thomson. 2015. Hypertext Transfer Protocol Version 2 (HTTP/2). IETF RFC 7540. 1–96 pages.

[3] Janardhan Iyengar and Ian Swett. 2015. QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2. Technical Report. Network Working Group. 1–30 pages.

[4] Nigel Williams, Sebastian Zander, and Grenville Armitage. 2006. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. ACM SIGCOMM Computer Communication Review 36, 5 (2006), 5–16.

[5] Velan, P., Čermák, M., Čeleda, P., Drašar, M.: A survey of methods for encrypted traffic classification and analysis. International Journal of Network Management 25(5), 355–374 (2015).

[6] I. Goodfellow, Y. Bengio, en A. Courville, Deep Learning. The MIT Press, 2016.

[7] R. Szeliski, Computer Vision: Algorithms and Applications, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2010.

[8] M. D. Harris, Introduction to Natural Language Processing. USA: Reston Publishing Co., 1985.

[9] Liu Q., Wu Y. (2012) Supervised Learning. In: Seel N.M. (eds) Encyclopedia of the Sciences of Learning. Springer, Boston, MA. https://doi.org/10.1007/978-1-4419-1428-6_451

[10] L. Ericsson, H. Gouk, C. C. Loy, en T. M. Hospedales, "Self-Supervised Representation Learning: Introduction, Advances and Challenges", arXiv [cs.LG]. 2021.

[11] T. Chen, S. Kornblith, M. Norouzi, and G. E. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," CoRR, vol. abs/2002.05709, 2020, [Online]. Available: https://arxiv.org/abs/2002.05709.

[12] J.-B. Grill et al., "Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning," CoRR, vol. abs/2006.07733, 2020, [Online]. Available: https://arxiv.org/abs/2006.07733.

[13] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, en R. Soricut, "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations", arXiv [cs.CL]. 2020.

[14] Rezaei, S., Liu, X.: Deep learning for encrypted traffic classification: An overview. IEEE communications magazine 57(5), 76–81 (2019).

[15] Zhitang Chen, Ke He, Jian Li, and Yanhui Geng. 2017. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In IEEE International Conference on Big Data (Big Data). 1271–1276.

[16] Jonas Höchst, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben. 2017. Unsupervised traffic flow classification using a neural autoencoder. In IEEE Conference on Local Computer Networks (LCN). 523–526.

[17] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2017. Automated website fingerprinting through deep learning. arXiv preprint arXiv:1708.06376 (2017).

[18] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. 2017. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In IEEE International Conference on Intelligence and Security Informatics (ISI). 43–48.

[19] Zhuang Zou, Jingguo Ge, Hongbo Zheng, Yulei Wu, Chunjing Han, and Zhongjiang Yao. 2018. Encrypted traffic classification with a convolutional long short-term memory neural network. In IEEE International Conference on High Performance Computing and Communications; IEEE International Conference on Smart City; IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS). 329–334.

[20] I. Akbari et al., "A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web," Proc. ACM Meas. Anal. Comput. Syst., vol. 5, no. 1, Feb. 2021, doi: 10.1145/3447382.

[21] A. Shahraki, M. Abbasi, A. Taherkordi, and M. Kaosar, "Internet Traffic Classification Using an Ensemble of Deep Convolutional Neural Networks," in Proceedings of the 4th FlexNets Workshop on Flexible Networks Artificial Intelligence Supported Network Flexibility and Agility, New York, NY, USA: Association for Computing Machinery, 2021, pp. 38–43. [Online]. Available: https://doi.org/10.1145/3472735.3473386

[22] A. Vaswani et al., "Attention Is All You Need," CoRR, vol. abs/1706.03762, 2017, [Online]. Available: http://arxiv.org/abs/1706.03762

[23] M. Liu et al., "Gated Transformer Networks for Multivariate Time Series Classification," CoRR, vol. abs/2103.14438, 2021, [Online]. Available: https://arxiv.org/abs/2103.14438

[24] A. S. Iliyasu and H. Deng, "Semi-Supervised Encrypted Traffic Classification With Deep Convolutional Generative Adversarial Networks," in IEEE Access, vol. 8, pp. 118-126, 2020, doi: 10.1109/ACCESS.2019.2962106.

[25] H. Fan, F. Zhang, R. Wang, X. Huang and Z. Li, "Semi-Supervised Time Series Classification by Temporal Relation Prediction," ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021, pp. 3545-3549, doi: 10.1109/ICASSP39728.2021.9413883.

[26] K. He, X. Zhang, S. Ren, en J. Sun, "Deep Residual Learning for Image Recognition", arXiv [cs.CV]. 2015.

[27] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of Encrypted and VPN Traffic using Time-related Features:," in Proceedings of the 2^{nd} International Conference on Information Systems Security and Privacy, Rome, Italy, 2016, pp.

[28] Pkt2flow. Accessed December 2021. [Online] Available: https://github.com/caesar0301/pkt2flow

[29] D. P. Kingma en J. Ba, "Adam: A Method for Stochastic Optimization", arXiv [cs.LG]. 2017.

[30] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python", Journal of Machine Learning Research, vol 12, bll 2825–2830, 2011.

[31] W. Falcon en K. Cho, "A Framework For Contrastive Self-Supervised Learning And Designing A New Approach", arXiv preprint arXiv:2009.00104, 2020.

[32] Dataset, W.V.: (2013), Accessed January 2022. [Online] Available: https://wand.net.nz/wits