2022 IEEE 8th International Conference on Network Softwarization (NetSoft) | 978-1-6654-0694-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/NetSoft54395.2022.9844082

Encrypted Network Traffic Classification in SDN 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

***Abstract*—Network traffic classification has a huge application in software-defined networking (SDN) where we talk about more control over the network traffic. With the increase of encrypted protocols in the network, the problem of traffic classification has become extremely challenging. Many researchers have proposed different techniques to do traffic classification. This demo paper presents an application of our proposed method for traffic classification in an SDN environment. The proposed method leverages one of the self-supervised learning approaches, an emerging field of deep learning, to classify network traffic. This paper shows that the proposed method can outperform the corresponding supervised approach by ∼2% in terms of accuracy using data collected from an SDN testbed. Furthermore, an SDN application is developed to show that the trained model is able to classify real-time traffic.**

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

I. Introduction

The modern networks are becoming complex day by day to support the heterogeneous needs of the consumers. To manage these complex networks automation is necessary. Therefore, modern networks are shifting towards softwarization by using technology like Network Function Virtualization (NFV) and Software-Defined Networking (SDN) [1]. Classification of network traffic can enable multiple applications in softwarized network, such as anomaly detection, traffic engineering, and intrusion prevention.

The most successful methods for traffic classification are from the supervised learning paradigm [2]. The problem with supervised learning is that it requires an ample amount of labeled data to train a model with good accuracy. Preparing this labeled data for network traffic is expensive. Furthermore, errors can happen during the preparation of this labeled data which can affect the accuracy of the trained model [3]. In another work, we have shown that Self-Supervised Learning (SSL) can outperform supervised learning when a small amount of labeled data are available [4]. The goal of this demo paper is to show an application of the proposed SSL technique that can classify real-time traffic in an SDN environment. While in [4] the model is trained using publicly available traffic classification data, in this paper, the data are generated from an SDN testbed.

SDN facilitates centralized control over network traffic by decoupling the data plane from the control plane. Therefore, to demonstrate the network traffic classification (NTC) ability of the SSL model proposed in [4], SDN is the appropriate choice. In this demo paper, an SDN environment is created using Mininet [5]. Then, some application-level traffic is generated inside the SDN testbed using the D-ITG tool [6]. We developed an application using RYU controller [7] to collect features from the testbed pertaining to the generated traffic. After collecting features from the testbed, an SSL model is trained according to [4]. The performance of the SSL model is also compared with a supervised learning model trained with the same labeled data. Our application is integrated with the trained SSL model to classify real-time traffic. All these steps including the obtained result are described in the remainder of this paper. Our evaluation shows that the proposed SSL model outperforms the supervised learning approach by ∼2% using data collected from the SDN testbed and the application can classify real-time traffic with high accuracy.

II. System Description

An application is developed using RYU controller that uses the trained model to classify network traffic. The same application is used to generate features for training the SSL model. The application sends requests to get flow statistics using southbound API (OpenFlow [8]) every second to the data plane and extracts features from the reply message. Then the features are given to the SSL model to get the classification. Figure 1 shows the topology of the network prepared for this demonstration. The process of feature collection from generated traffic to prepare training dataset is discussed in the following sections.

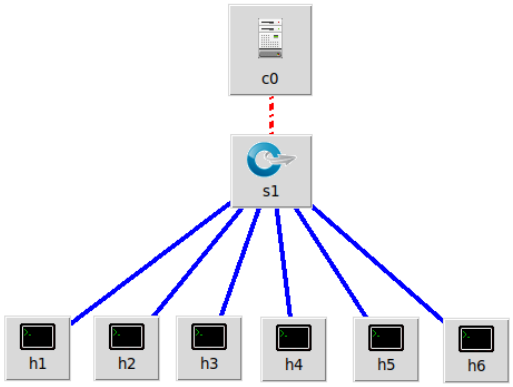

Fig. 1: Network Topology used in the demonstration

A. Traffic generation

At first, a network topology is created using Mininet that consists of an open vSwitch, six hosts, and a controller. The hosts are mentioned as h1, h2, h3, h4, h5, and h6 in the rest of the paper. The ITGSend module of the D-ITG tool is used to generate five types of application-level traffic from h1 in the testbed as shown in Table I. Due to the limitation of the D-ITG tool, we are restricted to use these five types of application-level traffic. The ITGRecv module of the D-ITG tool is started in every host from h2 to h6 as these hosts are acting as a receiver of the generated traffic. A script is written to generate 100 flows per class of traffic. The script takes a class name as input. Then, the destination host is selected randomly between h2 to h6. The traffic generation time is selected from a normal distribution whose mean is 1.67 minutes and the standard deviation is 0.15. All the flows are generated from h1 as the ITGSend module of the D-ITG tool is needed to be run from inside the host. Note that, although we are fixing the source, the trained model will be able to classify traffic between any host in the network. That is because the collected features will not change depending on the source or destination.

TABLE I: Generated traffic type

Traffic Type	Class Name
Active player in Counter Strike game	CSa
Idle player in Counter Strike game	CSi
Quake III game	Quake3
DNS traffic	DNS
VoIP traffic	VoIP

B. Feature collection

The developed RYU application in this demo can run in two modes. Firstly, in the feature collection mode, it can collect features and save them in a CSV file format. Secondly, in traffic classification mode, it can extract features and classify the ongoing traffic. While the data generation script is running inside Mininet, the application performs the following steps in feature collection mode:

- Set appropriate table entry to the flow table of the switch.
- Send flow statistics request message to the switch every second.
- Extract features from the reply message. The extracted features are packet and byte count in the forward and reverse direction in the last one second. These features are selected because firstly, they are easy to collect. In addition, these features are protocol agnostic. Therefore, the same features will work if encrypted protocol e.g., QUIC [9] is used by the application.
- Save all the features in a CSV file. This filename is decided by applying a hash function to the unique attributes (source and destination IP addresses and port numbers) of the flow.

C. Dataset preparation

According to our proposed method in [4], two sets of data are needed for training. One large unlabeled dataset for the pre-training stage and one small labeled dataset for the fine-tuning stage. Among the generated 100 flows per class, 70 flows per class are used to prepare an unlabeled dataset and the rest of the 30 flows per class are used to prepare a labeled dataset.

TABLE II: Number of sub-flows in unlabeled dataset

Class Name	Number of Sub-flows
CSa	1147
CSi	1548
Quake3	1485
DNS	2515
VoIP	1546

TABLE III: Number of sub-flows in labeled dataset

Class Name	Number of Sub-flows	Training set	Test set
CSa	517	103	414
CSi	698	139	559
Quake3	670	134	536
DNS	1135	227	908
VoIP	698	140	558

After generating the flows, the next step is to divide each flow into sub-flows according to [4]. For that, an incremental sampling technique is used which is described in [4]. These sub-flows are then used for the training. The number of sub-flows per class in the unlabeled dataset and labeled dataset are shown in Table II and III, respectively. The number of sub-flows per class depends on the selected traffic generation time from the normal distribution. Note that no data balancing operation is performed during the training phase to reflect on the real network scenario. For the labeled dataset, after creating sub-flows, 20% of the sub-flows are used for training, and 80% of the sub-flows are used for testing the model. This train-test split is mentioned in Table III as Training set and Test set. For the unlabeled dataset, the unique id (created in the previous step) is associated with the sub-flows. These unique ids will be used during the training to identify sub-flows created from the same flow. After preparing the dataset, training is done according to [4].

TABLE IV: Result on test dataset

Metric	Self-supervised method(%)	Supervised method(%)
Accuracy	**97.642**	95.382
F1	**97.342**	94.558
Recall	**97.144**	94.018
Precision	**97.562**	95.506

III. Evaluation on test dataset

The result of the SSL approach, in comparison with the supervised learning approach, is shown in Table IV. To get a stable result, the same training is run five times and the average results on the test dataset are shown in Table IV. The supervised model is trained using the same labeled dataset shown in Table III. From Table IV, it is clear that the proposed method outperforms the supervised learning approach by $\sim$2%.

TABLE V: Real-time classification result of the trained model

Test No.	Accuracy(%)	Precision(%)	Recall(%)	F1(%)	Avg. Inference Time(s)	Source	Destination
1	99.28	98.72	99.31	98.99	0.00207	h1	Random between h2 to h6
2	97.96	98.49	98.02	98.22	0.0025	h2	Random between h1 and h3 to h6
3	99.05	98.97	98.42	98.68	0.00215	h3	Random between h1, h2 and h4 to h6
4	98.66	98.85	98.83	98.81	0.00232	h4	Random between h1 to h3 and h5, h6
5	99.21	99.04	99.11	99.07	0.00228	h5	Random between h1 to h4 and h6

IV. Demonstration

Once the SSL model is trained, the trained model is then used by our RYU application for real-time traffic classification. The application runs in classification mode in this step. The workflow of the application in classification mode is shown in Figure 2. The SSL model is trained on sub-flows which are a portion of the actual flow. Therefore, during the real-time classification, the application classifies each sub-flow from a flow. For example, if a flow is generated from h1 to h3 for 300 seconds but is divided into 6 sub-flows during the training (each sub-flow is 45 seconds), then the application will classify the whole flow of 300 seconds 6 times. As a result, we will see six outputs for the same flow from the application.

To test the real-time classification ability of the trained model, five tests are done on the testbed. In each of the tests, twenty flows are generated with random hosts as destinations. The duration of each flow is selected from the same normal distribution described in section IIA. The output of the application is then saved in a log file. Later, a script is used to calculate real-time classification accuracy (shown in Table V) by comparing the log file and the traffic generation script. In this real-time evaluation of the trained model, the evaluation metrics are calculated by considering each sub-flow classification as an individual flow. All the scripts along with the dataset used for this demonstration are made available in [10]. During the traffic generation, a flow is generated to a specific port of the destination host. Therefore, we can verify whether the predicted class for a specific port matches the traffic generation script or not. For real-time traffic classification, it is important to know how long the model takes to classify a sub-flow. This time is shown as inference time (in second) in Table V. Note that in real-time evaluation, the source host is changed in every test. As expected, the accuracy is similar to the test dataset.

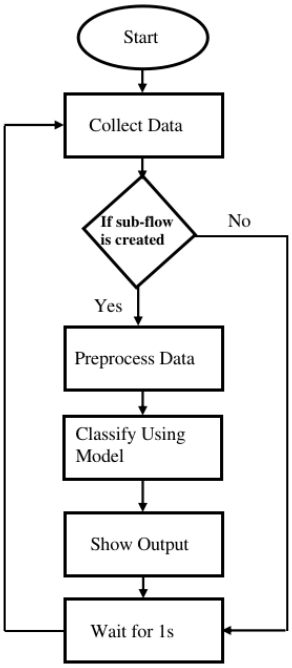

Fig. 2: Workflow of the RYU application in classification mode

V. Conclusion

This demonstration shows the application of SSL for network traffic classification in an SDN environment. From data collection to feature generation and finally traffic classification in real-time is demonstrated in this paper. Hopefully, this paper will encourage researchers to develop and evaluate more sophisticated ways of traffic classification in an SDN environment. In our future work, the overhead introduced in the network by the application will be investigated.

References

[1] F. Z. Yousaf, M. Bredel, S. Schaller and F. Schneider, "NFV and SDN—Key Technology Enablers for 5G Networks," in IEEE Journal on Selected Areas in Communications, vol. 35, no. 11, pp. 2468-2478, Nov. 2017, doi: 10.1109/JSAC.2017.2760418.

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

[3] J. Frank, U. Rebbapragada, J. Bialas, T. Oommen, and T. Havens, "Effect of Label Noise on the Machine-Learned Classification of Earthquake Damage," Remote Sensing, vol. 9, no. 8, p. 803, Aug. 2017, doi: 10.3390/rs9080803.

[4] Towhid, M. S., Shahriar, N.. Encrypted network traffic classification using self-supervised learning. In IEEE 8th International Conference on Network Softwarization (NetSoft), 2022. [To Appear].

[5] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, 2010.

[6] S. Avallone, S. Guadagno, D. Emma, A. Pescape and G. Ventre, D-ITG distributed Internet traffic generator, First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings., 2004, pp. 316-317, doi: 10.1109/QEST.2004.1348045.

[7] Kubo, R., Fujita, T., Agawa, Y. and Suzuki, H., 2014. Ryu SDN Framework—Open-source SDN Platform Software — NTT Technical Review. [online]. Available: https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201408fa4.html [Accessed 10 April 2022].

[8] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J., 2008. OpenFlow. ACM SIGCOMM Computer Communication Review, 38(2), pp.69-74.

[9] 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.

[10] Encrypted network traffic classification in SDN repository on Github. [Online]. Available: https://github.com/shamimtowhid/encrypted_network_traffic_classification_in_SDN.