Prediction of Recurrent Flow Characteristics Using Time Scale Ensemble Learning

Mike Petersen
Student-ID: 1221772
Fulda University of Applied Sciences
Email: mike.petersen@informatik.hs-fulda.de

Oliver Kovarna
Student-ID: 539151
Fulda University of Applied Sciences
Email: oliver.kovarna@informatik.hs-fulda.de

***Abstract*—The exponential growth of Internet traffic requires increasingly efficient resource utilization within network systems by providing both integrative and fine-grained management of network components. Recent advancements combine software-defined networking concepts and machine learning approaches in order to analyze network traffic behavior and enable automatic derivation of network and service management decisions, such as dynamic path determination based on predictions of network characteristics. A key challenge in classifying network flows is the adaptation to unforeseen changes in the underlying distribution of the data stream. This phenomenon, known as concept drift, leads to the degradation of deployed models resulting in poor predictions and decision outcomes. In this context, we present a network traffic classification method based on online ensemble learning. Specifically, a flow data stream is partitioned based on a set of windows with different time scales. Within the life cycle of each time window, both inference and training of a DNN is performed in an online fashion. The scores from the respective time windows are consolidated using an ensemble strategy. In this way, temporal dependencies between data instances are addressed and concept drifts of different types and speeds should be handled. Experiments on real network data demonstrate the accuracy performance of our approach in comparison to a single online model.**

***Index Terms*—Traffic Engineering, Flow Prediction, Machine Learning, Ensemble Learning, Concept Drift**

I. Introduction

A. Problem Statement

In the context of *network flow prediction*, machine learning approaches have recently emerged that enable accurate classification of network flow data and thus better management of network components and the incoming traffic. In many areas of the internet, providers of various services are challenged with a large volume of streaming data due to increasing link speeds and number of users. Since many unforeseen changes can occur in network traffic, it is necessary to process the data in an online fashion in order to detect drifts quickly and deal with them accordingly [1]. In general, unforeseen changes in data patterns are known as *concept drift* [1]. Considering the different types of concept drift is important for further investigation. A sudden drift occurs after a short period of time and causes a direct change in the data distribution. With a gradual drift, the new concept replaces the old concept linearly. An incremental drift occurs step by step over a certain period of time. When an online model encounters one of these three types it should adapt to these concepts and try to restore the previous accuracy. In case of *recurrent concepts*, a new concept is established only temporarily before restoring the old concept again. The different types of concept drift can happen at a certain point in time or can appear over a certain amount of time. Moreover, the concepts of these different types can transition into each other [2]. Deep Neural Networks (DNNs) usually are well qualified for the application within high volume data streams with drifts, because their architecture inherently allows an online classification. However, their online learning capability comes at cost as they suffer from a phenomenon called catastrophic forgetting (CF). The problem of CF refers to the complete elimination of previously learned information by a neural network, when exposed to new information [3], [4]. If the changes in the data statistics are recurrent drifts of high dynamic and intensity the model's performance can be degraded dramatically. The naive approach to cope with this challenge is to retain data in order to feed the network with old patterns that may get lost as the drift occurs. We assume that one way to address this problem is to support an online model with multiple offline models, each of which is based on an state from past time windows [5]. The fact that ensemble learners are used for dealing with class imbalance or concept drift [6] supports the decision to organize such different models in form of an ensemble. In this way, the above-mentioned short-term unforeseen changes within the network traffic shall be adequately addressed by implementing a robust continuous adaptation for the model.

B. Contributions

- We provide an overview of existing classification techniques that leverage the concept of multiple windows and highlight the differences with our approach.
- We perform online multi-class training and inference of a flow's bit rate by using an ensemble of DNNs in a streaming setup, where the state of each level 0 model is based on a different sized time window.
- We compare the accuracy performance between a single online classifier and the ensemble on real network data.
- We provide the code for the implementation of the presented architecture.

II. Related Work

A. Traffic Classification with Machine Learning

A comprehensive survey on the application of different machine learning paradigms and techniques to fundamental problems in networking is provided in [7]. An overview that focuses on the subject of network traffic control, specifically discussing deep learning based routing in contrast to conventional routing, is given in [8].

Poupart et al. [9] explore the application of different machine learning algorithms for network traffic prediction. By categorizing network traffic flows based on their predicted size into elephant and mice flows, the completion time of elephant flows is intended to be minimized. Seven features (source IP, destination IP, source port, destination port, protocol, server vs. client and size of the first three packets) are considered as input for the selected algorithms, which include an Online Bayesian Moment Matching (oBMM), a Gaussian Process Regression (GPR) and a Neural Network (NN). According to the classification results, mice flows are routed by equal cost multipath (ECMP) routing, while elephant flows are routed by least congested (LCR) routing. Consequently, the algorithms improve routing and scheduling, since flow sizes are already known at the beginning of a flow's life cycle. However, the runtime complexity of GPR and oBMM do not enable a scaled application in real network environments.

In [10], Malik et al. have developed a SDN-based flow routing framework, acting as the application layer for network traffic classification via a northbound API. Processes within this architecture are divided into two phases. During the learning phase, network traffic flows are collected periodically and stored in a historical data store. A flow pattern analysis and feature extraction constructs a refined database, that is utilized for the training and evaluation of a deep learning classifier. The subsequent action phase attains and utilizes learned information for load balancing and routing. Finally, the SDN controller uses the learned knowledge for routing updates, optimising the network's performance and classifying the network traffic. The proposed framework is tested on the Moore dataset by classifying the network traffic into distinct application classes, showing effective and scalable classification results.

Hardegen et al. [11] propose a processing pipeline for a network-flow based throughput classification using deep neural networks (DNNs). Flows are categorized into three classes based on their predicted bitrate. For this purpose, the feature set containing the standard 5-tuple is additionally enriched with metadata (e.g. private/public prefixes, VLANs, ASNs) extracted from the internal and global topology. Besides the presentation of the classification pipeline, an in-depth analysis of the characteristics of the real-world network traffic data collected for the experiments is performed. The evaluations show that a multi-class bit rate prediction for streamed flow data is feasible with DNNs and that enriching the flows can contribute to a significant improvement in accuracy.

B. Multiple Window Approaches

Li et al. [12] introduce a windowing approach that is intended to improve the classification of imbalanced streaming data. The different windows represent the current batch of instances, the latest minority instances, and the ensemble classifier. By keeping a window of minority instances, the algorithm copes with imbalanced data streams. For the current batches, a sliding window approach is applied. The optimal window size is determined heuristically. Before classification, the weights of each sub-classifier are adjusted by calculating similarities of the current and last window used to train each sub-classifier. The classification is performed by a weighed majority voting strategy. Training of a new sub-classifier is performed as soon as the current ensemble classifier is imprecise concerning minority and majority classes.

The network intrusion detection system Kitsune [13] performs an anomaly detection by using multiple autoencoders on a bagged feature space. The utilization of damped windows that incrementally update the systems's statistics saves memory and runtime. This way, only the most recent occurrence within the packet channel is recorded. The framework includes a packet capturer, a packet parser, a feature extractor, a feature mapper and the anomaly detector KitNET. The feature vector is split into groups of attributes, which are each mapped to an autoencoder within the ensemble. The root mean squared error (RMSE) between the original instance and the reconstruction that is calculated by each autoencoder is used as input for a subsequent autoencoder that calculates the final anomaly score.

Lazarescu et al. [14] present an incremental learning algorithm in that predicts and adapts to real and drifts using three different windows. While the smallest window deals with fast changing concepts, the largest window considers the slowest changing concepts. The input is labelled with a concept size and assigned to one of the windows. The small window's concept changes are used to determine the size changes of the other windows. The algorithm allows multiple hypotheses on the basis of recognised concepts and handles virtual drifts well. However, each concept requires an extra window, resulting in a high computational complexity.

C. Key Distinguishing Features

- We perform a supervised complete flow traffic classification in a true online fashion instead of splitting the input flows into subsets of train and test flows.
- The evaluation is conducted on a real world network traffic dataset that contains realistic network characteristics instead of using a synthetic dataset.
- Each window within the proposed architecture is provided with the full feature set but differ in the context of their timely state.
- Instead of explicitly detecting drifts in the data stream, the approach shall handle recurrent concepts by learning dependencies between the different windows by combining their results in a meta learner.

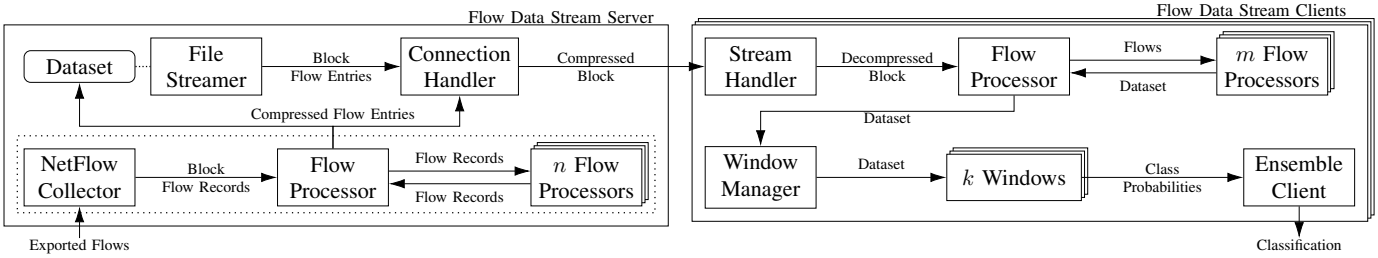


Fig. 1. The high level system architecture of the flow processing pipeline depicting the data flow between the main components.

III. Flow Data Processing Pipeline

First, the high-level system architecture describing the data flow between the individual components is outlined (see Figure 1). This is followed by a specification of the introduced components and their corresponding tasks. Thereby, the main system components from [11], on which our implementation is largely based, are covered. Finally, the ensemble extensions are described. The implementation of this architecture is publicly available in a Git repository[1].

A. Data Flow

Exported flows are collected by the *NetFlow Collector* until a defined block size (default 100 000) is attained. The *Flow Processor* then splits the block of flow records into disjoint subsets of equal size depending on the number of parallel processes being available. Each subset is passed to one of n parallel *Flow Processors*, which execute *Aggregation*, *Enrichment*, and *Anonymization* functions. The subsets are then merged back into a block and compressed by the *Flow Processor* and either passed to the *Connection Handler* for further processing or exported as an offline dataset. The exported data can be replayed by a *File Streamer* for reproducible experiments.

Multiple Flow Data Stream Clients can connect to a Flow Data Stream Server to receive data. A *Stream Handler* ingests the incoming data blocks, decompresses them, and forwards them to a *Flow Processor*. Analogous to the procedure in the Flow Data Streaming Server, the blocks are split and processed in parallel by m *Flow processors*. Feature selection, normalization and labeling functions are applied in each instance. The results are then reconstructed into one block again.

As flows are not emitted in a continuously sorted fashion regarding their timestamps due to the timeout configuration of the utilized flow exporters, incoming flows need to be organized for the subsequent creation of the time windows. Therefore, flows are first retained in a buffer in the *Flow Processor* until the effect of the export timeouts has been compensated for. Subsequently, the flows are sorted according to their timestamps and passed to the *Window Manager*.

The *Window Manager* distributes incoming flow blocks to k parallel *Windows*, thereby copying a block k times. Within each *Window*, a DNN is first tested and then trained on a complete flow block in a true online learning fashion. After a block has been classified multiple times by k *Windows* (Level 0 Classifiers), a barrier synchronizes the resulting class probabilities, which are then passed to the *Ensemble Client*. Here, the class probabilities are combined into a single input vector and fed into another DNN (Level 1 Classifier), whose output represents the final classification result.

[1]https://github.com/mikepetersyn/Flow-Data-Streaming-Client-Ensemble

B. Data Preprocessing

1) Flow Data Stream Server:

a) Aggregation: Due to configuration settings (timeouts) or hardware limitations (cache sizes) of a flow exporter, exported flow records may not describe a complete communication. Thus, flow records are aggregated block-wise based on their 5-tuple, timestamp and their flags.

b) Enrichment: Each flow entry is enriched with additional metadata containing information from local and global network contexts (e.g. private/public prefixes, VLANs, ASNs).

c) Anonymization: The anonymization of addresses is done by a function that substitutes address octets using substitution tables, which allow a preservation of relationships between IP addresses and their subnets while ensuring data privacy.

2) Flow Data Stream Clients:

a) Flow Filtering and Feature Selection: The feature-based filtering of the data reduces the overall number of flows in the stream. For instance, in order to keep only TCP traffic, flows with other transport protocols are dropped. The selection of a subset of features from the set of available features allow experiments with different feature combinations.

b) Labeling: A flow property selected for prediction is called label. Each flow entry is assigned to a class by mapping the selected flow property to a class using predefined boundaries. The boundaries for predicting the bit rate ($^{bit}/_{sec}$) are selected as: class $0 = [0, 50[$, class $1 = [50, 8000[$, class $2 = [8000, \infty]$.

c) Normalization: The values of dynamically chosen ports are replaced by zero values and timestamps are split into their different units. The normalization transforms a feature into a different format (float, bit pattern or one-hot). Table I gives an overview of all available features in the data stream with corresponding normalization formats and the respective size of the output vector. Values in bold represent the selected format for the experiments.

C. Flow Timeout Interference

Training DNNs based on different time scales requires the management of time windows, which in turn expect the arrival of time-sorted data. In this context, the behavior of a flow exporter is addressed and potential solutions are discussed.

1) Flow Exporter Behaviour: Due to the selected timeout values of a flow exporter, the timestamps of individual flows are not sorted as they arrive at the flow collector. A flow record is exported if a network communication is either inactive for t_i seconds or the communication duration exceeds t_a seconds. Considering a flow f_0 with a timestamp t_{f_0}, there is a chance that flows f_i appear subsequently with timestamps $t_{f_i} < t_{f_0}$ until the point of time $t_{f_0} + \max(t_i, t_a)$. Although the flow timeout settings allow a trade-off between exporting as early as possible with a high volume of data and exporting relatively late with a low volume of data, however, the unsorted emission of flows is in the nature of the process and can only be readjusted by downstream control mechanisms.

2) Data Stream Semantics: In practice, there are basically two different approaches to process potentially infinite data streams with a fixed amount of memory and without random access to the data. First, there are compression techniques that attempt to summarize the data (e.g. by a continuous calculation of the average w.r.t a certain attribute). Second, there are windowing techniques that attempt to portion the data into finite pieces. The first approach is not reasonably applicable to our problem. Windowing techniques, on the other hand, would be mandatory in a production system, but are complicated to implement. Thus, for an experimental evaluation, a less complex buffering method that simulates a windowing mechanism was chosen to prepare the data.

3) Flow Buffer: Within a block, flows are sorted by their respective timestamp due to the processing semantics in the flow processing server. This means that only the blocks in the stream have to be considered. For this, the timestamp t_{0_b} of the first flow in the block is set as the start of the buffer. Incoming blocks are held until a block arrives whose first flow has a timestamp $t_{0_{b+n}}$ such that $t_{0_{b+n}} - t_{0_b} >= d$. Consequently, the buffer size d must be greater than or equal $\max(t_i, t_a)$ to compensate for the timeout behavior. After a sufficient number of flows have accumulated in the buffer, these flows are sorted w.r.t their timestamps and passed to the *Window Manager*.

D. Windows

The *Window Manager* handles the opening and closing of k *Windows* by keeping record of the timestamps and sending the respective signals via a queue. The duration between the opening and closing is referred to as window iteration and determined by the window size. Two consecutive window iterations are called a cycle. Within the first half of a cycle, the corresponding online classifier first performs an inference phase and then a training phase per incoming block. In the second half of the cycle, the model enters an offline mode, so that only inference is performed and the state of the model remains unchanged for one window iteration (time scaled).

TABLE I
FEATURES IN THE DATA STREAM

	Feature	Format		
		Float	Bit	One-Hot
Level 0 Classifier				
Time and 5-tuple Information from Data Collection				
	month	**1**	4	12
	day	**1**	5	31
	hour	**1**	5	24
	minute	**1**	6	60
	second	**1**	6	60
	protocol	1	**8**	✗
	IP address	4	**32**	✗
	port	1	**16**	✗
Network Context from Data Enrichment				
	network	4	**32**	✗
	prefix length	1	**5**	✗
	ASN	1	**16**	✗
	longitude	**1**	✗	✗
	latitude	**1**	✗	✗
	country code	**1**	8	240
	VLAN	1	**12**	✗
	locality	1	**1**	2
Level 1 Classifier				
Decision Context from Level 0 Classification				
	class prob 1	**3**	24	✗
	class prob 2	**3**	24	✗
			...	
	class prob k	**3**	24	✗

E. Stacking Ensemble

The method of stacked generalization [15] aims at minimizing the error rate of one or more level 0 models, by combining their scores as input for a subsequent generalization in a second space, attempting to create a strategy for combining the output of its preceding models. In general, any number of levels can be stacked in this way, however, in practice this is limited by the rapid increase in complexity. The setup in this architecture specifies two levels (see Figure 2). In level 0, k different classifiers C, each having a different window size, process input vectors $\boldsymbol{x}$ and pass the resulting k class probability vectors $C_i(\boldsymbol{x})$ to the level 1 meta model $\boldsymbol{M}$. In general, stacking further divides the training portion of the dataset so that each level in the stacking model is assigned different data to train without leaking information regarding the target to the meta model. Since in the introduced architecture the models operate online, all data is fully processed at each level.

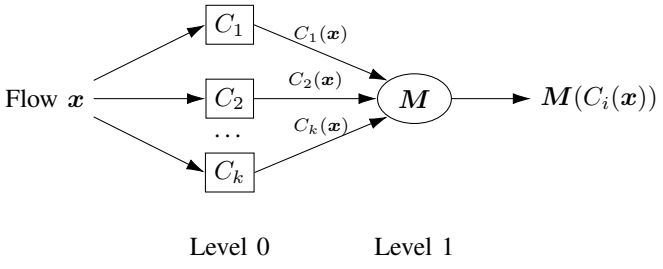


Fig. 2. Illustration of Stacked Generalization.

IV. Flow Prediction Experiments

In this section, details of the experimental setup and the results of the evaluation are presented. At first, the evaluation dataset, is examined in more detail. Then, the architecture and configuration of the hyperparameters of the utilized classifiers (DNNs) are described. Finally, the evaluation results are discussed.

A. Dataset

The network flow dataset from [5] was replayed with the *File Streamer* for the performance evaluation. The dataset contains bidirectional flow data that was collected by two core routers using the NetFlow protocol [16] in a real-world production network. In total, the dataset contains roughly 480 million flow entries that were continuously collected for seven days, beginning December 2, 2019 at 2:36 P.M. until December 9, 2019 at 2:36 P.M. (CET). For the following experiments, the first 72 hours only were selected. Both exporters used for collection were configured with an active timeout of 600s and an inactive timeout of 30s. Flow records were exported only for ingress traffic in order to avoid duplicated exports from one router. From the available set of features, the number of packets and bytes, the duration and the bit rate of a flow can be selected as possible class labels for a multi-class classification problem. As the prediction of the throughput remains the most challenging problem [5], the bit rate with class boundaries as described in Section III-B2 is selected as class label for the evaluation.

Looking at the distribution of class labels over the selected period (see Figure 3), it can be seen that in the morning, approximately just after 6:00 A.M. , until approximately 6:00 P.M. , there is a sharp increase in network traffic dynamics, such that the frequencies of the individual class labels fluctuate strongly during this period. In addition to the fluctuations themselves, it can also be seen that the overall distribution of classes is not balanced. To prepare the unbalanced data for model training, appropriate class weights are calculated for each block during the labeling process, which are used during training. However, compared to the calm period between 6:00 P.M. and 6:00 A.M. , the fluctuations suggest a higher variability of protocols and applications within the network traffic, which in turn is expected to result in a more difficult prediction and a decreasing accuracy, respectively.

B. Neural Network Architecture

1) Level 0: Each of the k level 0 classifier is a fully-connected DNN, that share the same hyperparameter configuration (see Table II) based on an optimization performed in [17]. The architecture consists of five hidden layers with 1 000 neurons each. On each layer (except the output layer) a dropout ($p = 1.0$) is applied. The training is performed mini-batch-wise (32) by minimizing standard cross-entropy loss with Adam Optimizer and a training rate of $1e{-}4$. Each incoming block is trained once only. A rectified linear unit transfer function is applied to the hidden layers. The output layer delivers class probabilities as output by utilizing a softmax function.

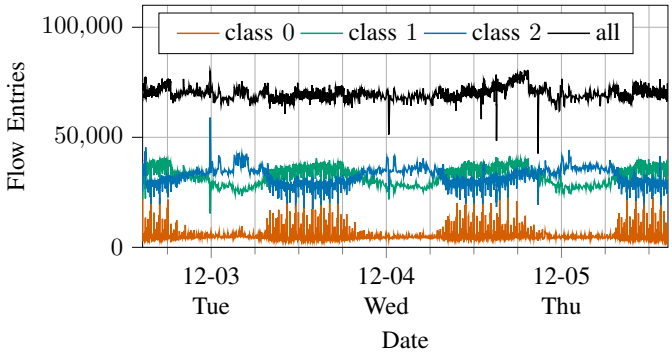


Fig. 3. Bit rate class distribution of flow entries for every block (100 000 flows) of the first three days of the dataset.

2) Level 1: Since the input size of the Level 1 classifier is less complex, only three hidden layers are chosen here instead of five. The remaining values for the configuration are identical to the previous level.

C. Time Scaled Ensemble Learning

1) Hardware: The hardware available for the experiments (see Table III) is not sufficiently performant to process the amount of data generated at midday in real time speed. Thus, in the experiments, new blocks were only made available as soon as the current ones had already been processed.

2) Comparability Parameters: Three DNNs with different window sizes (0h, 1.5h, 3h) are chosen as level 0 classifier. A window size of zero effectively describes a pure online model without any offline phases as described in Section III-D. The values for the window sizes were picked heuristically, since we are not aware of any comparable architecture or mathematical method for the specification of time windows. Regarding the evaluation frequency, the number of flow records per inference step and training epoch is set to 5 000. In addition, the number of epochs per block was fixed to 1 to get the results of a system with minimal hardware requirements and exclude the influence of a varying number of training iterations on the accuracy.

TABLE II
Hyperparameter Configuration

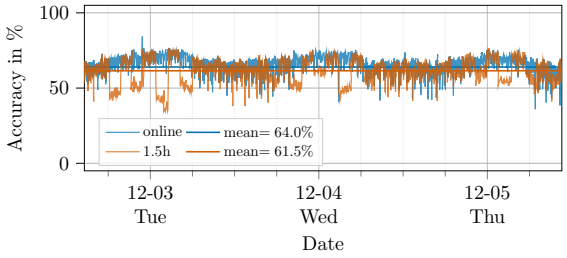
	Parameter	Level 0	Level 1
n_x	Input Size	243	$3k$ (9 with $k = 3$)
n_y	Output Size (No. Classes)	3	3
(p_i, p_h)	Dropout (input, hidden)	$(1.0, 1.0)$	$(1.0, 1.0)$
L	No. Hidden Layers	5	3
n_h^l	No. Neurons Layer l	1000	1000
η	Learning Rate	$1e{-}4$	$1e{-}4$
$\|B\|$	Batch Size	32	32
E_b	No. Epochs per Block	1	1

TABLE III
Employed Computational Resources

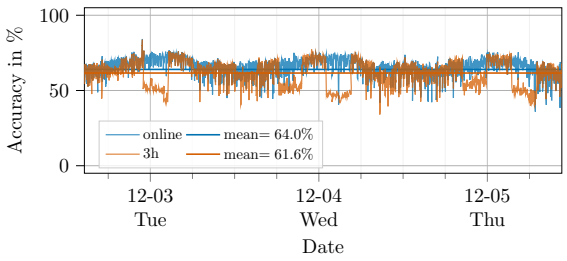
GPU Type	CUDA Cores	VRAM (GB)	Clocking (MHz)	CPU (# × GHz)	RAM (GB)
Quadro P1000M	640	4	1354	12 × 2.2	32

3) Experiment Analysis: Experiments show that an ensemble of different scaled time windows are able to support the classification accuracy of a single online learning model. However, the increase in accuracy with the configuration presented is insignificantly small, which leads to conclusion that the additional cost of operating an ensemble is not justified.

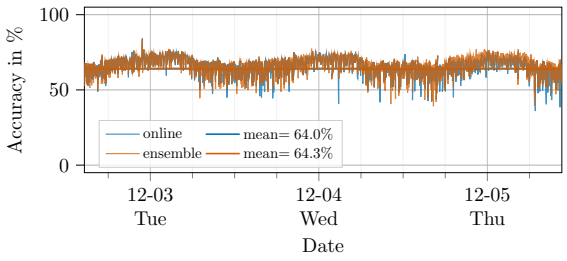
Figure 4 visualizes the performance accuracy comparison between the online model, the window models and the stacking ensemble, measured for each block of the dataset within the selected time period. It can be seen that the window models perform worse on average than the online model. An interesting observation is that significant performance drops can be observed for both window models as soon as they enter the offline classification phase. The offline learning phases may be ineffective due to the dynamics that the network traffic features. Particularly in the quiet phase between 6:00 P.M. and 6:00 A.M. , the accuracy of the classification drops to the level of a random process. However, these results contradict the assumption made earlier in Section IV-A about the dynamics at different times of the day. One reason for that could be that flows follow unique patterns at nighttime and more repetitive patterns occur at midday, even though a higher fluctuation of the label distribution is observed here. A detailed analysis of the data regarding the contained concepts, which takes into account the temporal dependencies, would provide insight at this point. Comparing the ensemble with the online model, a major difference is only noticeable from December 4 at 6:00 P.M. for a long phase, where the ensemble has roughly 5% higher accuracy on average. This may occur due to changes in the data distribution of the dataset, as the ensemble algorithm does not change within this particular phase.



(a) The 1.5h window size performance comparison.



(b) The 3h window size performance comparison.



(c) The stacking ensemble performance comparison.

Fig. 4. The mean accuracy measured for each incoming block of flows and the overall mean accuracy for all flows within the selected time period.

V. Conclusion and Future Work

At the beginning, a short overview of related work was given, in which on the one hand network flow prediction and on the other hand the utilization of a time-window concept for the improvement of machine learning algorithms was addressed. To the best of our knowledge, this work represents a novelty at the current time that combines the two topics thematically. For this purpose, an already existing architecture for flow prediction was extended such that, firstly, several windowed base learners can be operated in parallel and their results can be combined in a meta learner. Secondly, the classification is now performed in a true online manner, resulting in more application-oriented test outputs than in previous evaluations. For the presented architecture, the code was provided and the experimental framework parameters were described in detail to ensure the reproducibility of the results. Subsequently, the proposed approach was experimentally investigated by measuring performance differences in terms of prediction accuracy in the context of the bit rate of a flow. A subsequent analysis of the results showed that no significant improvements were achieved with the proposed approach in this particular configuration. Based on this, the following issues are considered for future work.

- Performing a dataset analysis, which examines concept changes over time, should reveal points of improvement in the algorithm.
- More extensive experiments with greater variation in window sizes, should provide better insight into the approach.
- Introducing a temporal aspect into the feature space of the Level 1 classifier, by using the flow timestamps, and processing it via a Recurrent Neural Network could recognize temporal dependencies between windows.
- An algorithm for dynamically determining window sizes based on the performance of individual level 0 classifiers would provide flexibility in adapting to the data stream.
- Besides the separation of the ensemble into different time scales, bagging the feature space in order to diversify the ensemble's base models could improve the general performance of the stacking ensemble.
- Focusing solely on improving the prediction of a flow's bit rate, which is the main challenge in this setup, a stacking ensemble based on level 0 classifiers, each operating on a different label, could be a promising concept. Thus, combining the results of the prediction of the number of packets, the number of bytes, and the duration of a flow could achieve an accurate bit rate prediction.

REFERENCES

[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, 2014.

[2] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *CoRR*, 2020.

[3] M. McCloskey and N. J. Cohen, "Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem," *Psychology of Learning and Motivation*, 1989.

[4] R. Ratcliff, "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions." *Psychological Review*, 1990.

[5] C. Hardegen, B. Pfulb, S. Rieger, and A. Gepperth, "Predicting Network Flow Characteristics Using Deep Learning and Real-World Network Traffic," *IEEE Trans. Netw. Serv. Manage.*, 2020.

[6] O. Sagi and L. Rokach, "Ensemble learning: A survey," *WIREs Data Mining and Knowledge Discovery*, 2018.

[7] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, 2018.

[8] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, 2017.

[9] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Yanhui Geng, Li Chen, K. Chen, and Hao Jin, "Online flow size prediction for improved network routing," *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, 2016.

[10] A. Malik, R. de Fréin, M. Al-Zeyadi, and J. Andreu-Perez, "Intelligent SDN Traffic Classification using Deep Learning: Deep-SDN," *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, 2020.

[11] C. Hardegen, B. Pfulb, S. Rieger, A. Gepperth, and S. Reismann, "Flow-based Throughput Prediction using Deep Learning and Real-World Network Traffic," *2019 15th International Conference on Network and Service Management (CNSM)*, 2019.

[12] H. Li, Y. Wang, H. Wang, and B. Zhou, "Multi-window based ensemble learning for classification of imbalanced streaming data," *World Wide Web*, 2017.

[13] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," *arXiv:1802.09089 [cs]*, 2018.

[14] M. M. Lazarescu, S. Venkatesh, and H. H. Bui, "Using multiple windows to track concept drift," *Intelligent data analysis*, 2004.

[15] D. H. Wolpert, "Stacked generalization," *Neural networks*, 1992.

[16] B. Claise, "Cisco Systems NetFlow Services Export Version 9," https://rfc-editor.org/rfc/rfc3954.txt, 2004, accessed: 2021-03-11.

[17] B. Pfülb, C. Hardegen, A. Gepperth, and S. Rieger, "A Study of Deep Learning for Network Traffic Data Forecasting," *arXiv:1909.04501 [cs]*, 2019.