FlowFormers: Transformer-based Models for Real-time Network Flow Classification

Rushi Babaria[§] Sharat Chandra Madanapalli[§] Himal Kumar Vijay Sivaraman
Computer Science *Electrical Engineering & Telecomms* *Canopus Networks* *Electrical Engineering & Telecomms*
BITS Pilani, India *UNSW Sydney, Australia* *Sydney, Australia* *UNSW Sydney, Australia*

***Abstract*—Internet Service Providers (ISPs) often perform network traffic classification (NTC) to dimension network bandwidth, forecast future demand, assure the quality of experience to users, and protect against network attacks. With the rapid growth in data rates and traffic encryption, classification has to increasingly rely on stochastic behavioral patterns inferred using deep learning (DL) techniques. The two key challenges arising pertain to (a) high-speed and fine-grained feature extraction, and (b) efficient learning of behavioural traffic patterns by DL models. To overcome these challenges, we propose a novel network behaviour representation called FlowPrint that extracts per-flow time-series byte and packet-length patterns, agnostic to packet content. FlowPrint extraction is real-time, fine-grained, and amenable for implementation at Terabit speeds in modern P4-programmable switches. We then develop FlowFormers, which use attention-based Transformer encoders to enhance FlowPrint representation and thereby outperform conventional DL models on NTC tasks such as application type and provider classification. Lastly, we implement and evaluate FlowPrint and FlowFormers on live university network traffic, and show that a 95% f1-score is achieved to classify popular application types within the first 10 seconds, going up to 97% within the first 30 seconds and achieve a 95+% f1-score to identify providers within video and conferencing streams.**

1. Introduction

Network traffic classification (NTC) is widely used by network operators for tasks including network dimensioning, capacity planning and forecasting, Quality of Experience (QoE) assurance, and network security monitoring. However, traditional classification methods based on deep packet inspection (DPI) are starting to fail as network traffic gets increasingly encrypted. Many web applications now use HTTPS (i.e. HTTP with TLS encryption) and browsers like Google Chrome now use HTTPS by default [1]. Applications like video streaming (live/on-demand) have migrated to use protocols like DASH and HLS on top of HTTPS. Non-HTTP applications which are predominately UDP-based real-time applications like Conferencing and Gameplay also use various encryption protocols like AES and Wireguard to protect the privacy of their users. With emerging protocols like TLS 1.3 encrypting server names, and HTTP/2 and QUIC enforcing encryption by default, NTC is bound to get even more challenging.

In recent years researchers have proposed to use Machine Learning (ML) and Deep Learning (DL) based models to perform various NTC tasks such as IoT device classification, network security, and service/application classification, ranging from coarse grain application type (e.g. video streaming, conferencing, downloads, gaming) to specific application providers (e.g. Netflix, YouTube, Zoom, Skype, Fortnite). However, many of these existing approaches train ML/DL models on byte sequences from the first few packets of the flow. While the approach of feeding in raw bytes to a DL model is appealing due to automatic feature extraction capabilities, it usually ends up learning patterns such as protocol headers in un-encrypted applications and server name in TLS based applications. Such models have failed to perform well in the absence of such attributes [2], for example in TLS 1.3 that encrypts the entire handshake thereby obfuscating the server name.

Our work takes an alternative approach by building a time-series behavioural profile (a.k.a. traffic shape) of the network flow, and using that to classify network traffic at both application type and provider level. Our **first** contribution §3 develops a method to extract flow traffic shape attributes (aka FlowPrint) at high-speed and in real-time. FlowPrint's data representation format keeps track of packet and byte counts in different packet-length bins without capturing any raw byte sequences, and provides a richer set of attributes than the simplistic byte and packet counting approach in our prior work [3]. It also operates in real-time, unlike other approaches e.g. [4] that perform post-facto analysis on packet captures. We show that FlowPrint is amenable for implementation in modern programmable hardware switches operating at multi-Terabit scale, and is hence suitable for deployment in large Tier-1 ISP networks.

Our **second** (and most significant) contribution §4 proposes FlowFormers: DL architectures that introduce attention-based transformer encoder [5] to the traditional Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM) networks. Transformer encoder greatly improves the performance as it allows the models to give attention to the relevant parts of the input vector in context of the NTC task. In other words, transformer encoder enhances our FlowPrint data prior to being fed to CNN and LSTM.

§. Equal contribution.

In our **final** contribution §5, we evaluate both FlowPrint and FlowFormers on a real-world dataset obtained from our university campus traffic. We evaluate the data representation and the models on NTC tasks identifying (1) Application type (e.g. Video vs. Conferencing vs. Download etc.), (2) Video provider (e.g Netflix vs. YouTube vs. Disney etc.) and (3) Conferencing provider (Zoom vs. MS Teams vs. Discord etc.). We show that using FlowPrint collected just for the first 10 seconds of a flow yields a 95+% f1 score to identify 5 types of applications. We further show that applying transformers increases the accuracy of both CNNs and LSTMs consistently across all the tasks. We demonstrate that the use of a transformer encoder together with LSTM (TE-LSTM) performs the NTC tasks with f1 scores 97.15%, 95.68% and 94.92% respectively.

2. Related Work

Network Traffic Classification (NTC) has been extensively studied by many researchers since a long time. Recent prior work for encrypted NTC using DL can be categorized into *packet-based*, *flow content-based* and *flow time-series-based*. Work in [6] and [7] classifies applications using 1-D CNNs and Stacked Auto Encoders (SAE) on byte sequences extracted using packet headers and/or payloads. *Flow content-based* approaches use RNN/LSTM models in addition to CNN and SAE with features collected over multiple packets within a flow like session bytes (concatenated packet payloads) [2], [8] packet inter-arrival times [9], and packet lengths to identify internet apps.

Most of the prior work in *packet-based* and *flow content-based* categories was evaluated using a public dataset [10] which contains many un-encrypted protocols such as SMTP, POP3 (email label) and SFTP and FTP (file transfer label) which are easily distinguishable using just the packet headers and/or first few payload bytes. Further, to classify encrypted applications/services based on HTTPS, DL-models often fit to features in TLS handhake such as cipher info and server name indication field (SNI) (e.g. youtube.com) without which the model accuracy drops significantly [2]. Thus, such approaches are either outdated, due to the increasing use of encryption or are susceptible to failure with upcoming protocols like TLS 1.3 wherein even the handshake is completely encrypted. Our work doesn't consider packet payloads as input but instead relies on flow's traffic shape and behaviour characteristics which are robust even with TLS 1.3 encrypted traffic.

Flow time-series-based approaches [3], [4], [11], [12], [13] rely purely on time-series features such as packet/PDU (protocol data unit) lengths [12], [13], [11] of a flow over time, inter-arrival times [11] and/or statistical features like transfer rates (bps/pps), burstiness, idle-time etc. derived from downstream bps [3]. The works [11] and [13] tackle IoT device classification and website fingerprinting tasks which are different in nature to application type and provider classification tasks that our work tackles. The scope of our prior work [3] was limited to Video vs. Download classification i.e. only two out of the 5 application types we tackle in this work. Authors in [4] use all of the above features i.e. time-series packet lengths, IATs along with TLS handshake bytes (excluding SNI and cipher info) and summarized flow statistics. While [4] comes the closest to our work, it's dataset contains only TLS/QUIC-based flows and it relies on statistical flow information which is only available at the end of the flow. In other words, it performs a post-hoc classification of the flow whereas we aim to perform real-time classification of the flow (within the first few seconds). Further, in addition to TLS and QUIC we also classify encrypted Conferencing and Gaming traffic from providers like Zoom, Skype, CS:GO, Call of Duty etc.

Our work is also a time-series-based approach which aims at popular NTC tasks which are based on flow's behaviour profile: application type and/or provider identification. We propose a new input format (FlowPrint) consisting of time-series counters to capture the traffic shape that is agnostic to protocol specific information often found easily in headers, handshakes or certificates. While it can be used with CNN/LSTM-based models, we additionally develop transformer-based model architectures which use attention mechanism to pick intricate traffic shape patterns from FlowPrint and outperform traditional DL models. To the best of our knowledge, our work is the first to propose the use of a transformer-based DL models in the context of NTC to identify internet applications/services.

3. FlowPrint

FlowPrint is a data format built using counters to capture the traffic shape and behavioural profile of network flows. The data captured in FlowPrint doesn't include header/payload contents of packets and hence is protocol-agnostic and doesn't rely on clear-text indicators like SNI. It aims to support wide range NTC tasks which rely on activity profile such as application type identification (e.g. Video vs. Conferencing vs. Download), application service detection (e.g. Netflix, Zoom etc.), Device Identification (IoT sensors, smart gadgets etc.) etc. FlowPrint has been designed to be implementable not just in software, but also in modern P4 programmable network switches like Intel Tofino [14] that operates at several terabit per sec.

FlowPrint consists of four 2-D arrays: upPackets, downPackets, upBytes and downBytes. Each array consists of two dimensions: (length bins, time slots). As shown in Fig. 1, an incoming packet is placed into appropriate bin i based on its length. A list of packet length boundaries (*PLB*) creates b discrete length bins (on the y-axis). On the x-axis, the packet is placed into the time slot j in which it arrives relative to the flow start – duration of the time slot is an input parameter called *interval*. Assuming its an upload packet, the cell (i,j) in upPackets array is incremented by 1 and the cell (i,j) in upBytes array incremented by the payload length of the packet (refer to Fig. 1). Thus, cell (i,j) of upPackets would contain the sum of all packets that arrive in time slot j with lengths between PLB[i-1] and PLB[i]. The process remains similar in the other direction – down arrays are shown stacked in dark shade.

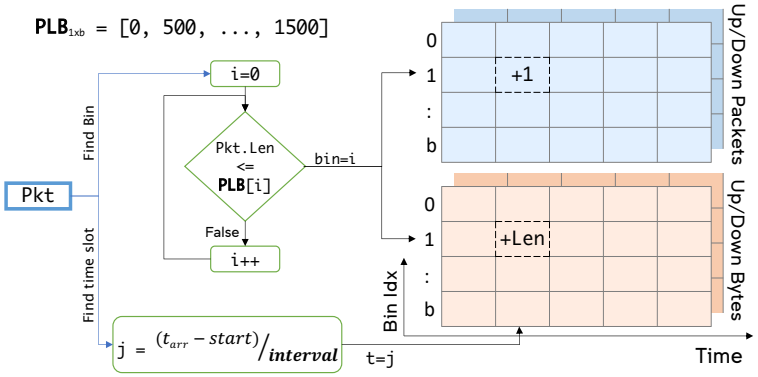

Figure 1: FlowPrint Datastructure and Algorithm

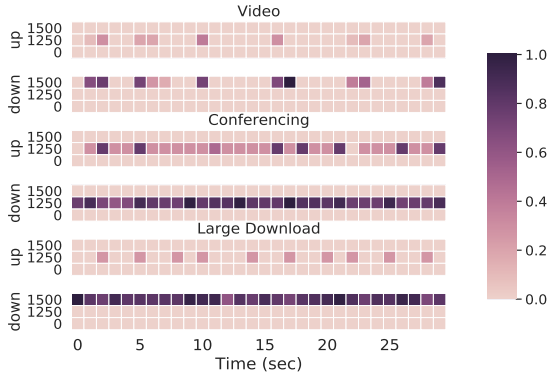

Figure 2: FlowPrint Examples (only Bytes shown)

The choice of *interval* and *PLB* determines FlowPrint's granularity and size. One may choose to have a small *interval* say 100ms and have 3 packet length boundaries or a large *interval* say 1 sec have 15 packet length bounds (in steps of 100Bytes). Such a choice needs to be made depending on the NTC task and compute/memory resources. We explore some of the trade-offs of FlowPrint configurations in §5.3.

Fig. 2 shows FlowPrint examples collected in our dataset. They show normalized upBytes and downBytes of 3 application types: Video, Conferencing and Large Download. The parameters used for the example are: $interval = 1sec$ and $PLB = [0, 1250, 1500]$ – intuitively these length boundaries attempt to form 3 logical bins: ACKs, MTU-sized packets and packets in between. One can observe that FlowPrint clearly demarcates the behavioural profile of the flows. The video flow on top shows periodic activity – there are media requests going in the up direction with payload length between 0 and 1250 and correspondingly media segments are being sent by the server using MTU-sized packets that fall in the bin (1250,1500]. Conferencing on the other hand is continuously active in the mid-bin (0,1250) in both upload and download direction with down being more active due to video transfer as opposed to audio transfer in the upload. A large download transferred typically using HTTP-chunked encoding involves the client requesting chunks of the file to the server which responds continuously with MTU-sized packets (in highest bin) until file downloads. Thus, this example illustrates the ability of FlowPrint to capture the traffic shape that can create markedly different patterns to identify application types.

We note that by a flow, we mean a set of packets identified using a flow_key constructed out of packet headers. Often, a 5-tuple consisting of srcip, dstip, srcport, dstport and protocol is used to form a flow_key to identify network flows at the transport level i.e. TCP connections and UDP streams. While our work also uses 5-tuple flow_key, FlowPrint is not inherently constrained by it. One may even use a 2-tuple (srcip and dstip) to construct a flow_key to identify all the traffic between a server and a client as a flow.

FlowPrint is amenable to implementation in high-speed P4 programmable switches [14]. A flow can be identified using its flow_key as match in a table, and sets of 4 registers can keep a track of up/down byte and packet counters. A controller can periodically poll the registers to get time-series of the counters at the defined *interval*. Once classified, the registers can be reused for a new flow.

FlowPrint fundamentally consists of four 2-D arrays. However, it can be extended by deriving two additional arrays: upPacketLength and downPacketLength by dividing the Bytes arrays by the Packets arrays in each direction. For instance, the upPacketLength[i,j] will contain the average packet lengths of packets which arrived in time slot j and are in the length bin i. These arrays can give precise time-series packet length measurements across the length bins. It is specifically useful to identify providers (e.g. Netflix, Disney) within a particular application type (video) as the overall shape remains very similar. In summary, the FlowPrint data structure has six 2-D arrays – collected over two directions and three counter types (packets, bytes, lengths) with each array of dimensions (numbins, time slots).

4. Transformer-based Classification

In the section, we develop transformer-based DL models which efficiently learn features from FlowPrint to perform NTC tasks. We first explain the specific NTC tasks we consider in our work and our dataset. We then provide a background on 1-D CNN and LSTM models which are commonly used DL model architectures for NTC tasks. We also explain the process to convert the FlowPrint arrays into suitable input formats for these models. Finally, we present a brief overview of transformer encoder and develop FlowFormer models by introducing transformer encoders to the CNN and LSTM architectures.

4.1. Objective & Dataset

In our work, we tackle two specific NTC tasks: (a) Application Type Classification: Identify the type of an application (e.g. Video vs. Conference vs. Download etc.) and (b) Application Provider Classification: Identify the provider of the application/service (e.g. Netflix vs. YouTube or Zoom vs. Microsoft Teams etc.). These tasks are typically performed today in the industry using traditional DPI solutions but however rely on information like DNS, SNI or IP-block/AS based mapping. Due to increasing encryption adoption these solutions may no longer work and hence we instead take an alternative approach focusing on using traffic behavioral profile captured by FlowPrint.

Table 1: Classification Dataset

Task	# Flows per class	Classes
Application Type	40,000	Video, Live Video, Gameplay, Conferencing and Downloads
Video Provider	30,000	Netflix, YouTube, DisneyPlus and PrimeVideo
Conference Provider	40,000	Zoom, Microsoft Teams, Whatsapp and Discord

Application Type Classification: The task identifies 5 common application types: Video streaming, Live video streaming, Conferencing, Gameplay and Downloads. An ML model will be trained to classify a flow into one of these 5 classes. Each type contains flows from different providers to make it diverse and not limited to provider-specific patterns. For instance, the Gameplay class has examples from the top 10 games active in our university network. For large downloads, while one may consider traffic from different sources, we chose Gaming Downloads/Updates from providers like Steam, Origin, Xbox and Playstation since they tend to be consistently large in size as opposed to downloads from other providers like Dropbox etc. which may contain smaller, say PDF, files. We note that Live video (video broadcasted live for example on platforms like Twitch etc.) has been intentionally separated from video on-demand to create a challenging task for the models.

Application Provider Classification: This task identifies the provider within each type of application. We choose two popular types: Video streaming and Conferencing (and correspondingly train separate models). The objective is to detect the provider serving that content type. For Video, we detect if it is one of Netflix, YouTube, DisneyPlus or PrimeVideo (top providers used in our university). For conferencing, we detect if it is one of Zoom, Microsoft Teams, WhatsApp or Discord – two popular video conferencing platforms and two popular audio conferencing platforms.

Dataset: To perform the tasks above, we need a labelled FlowPrint dataset. We obtained the labels from a third-party commercial DPI which associates both application type and provider to each flow being collected. So, every data record is a three tuple $< FlowPrint, Type, Provider >$. The FlowPrint arrays are recorded for 30 seconds at an *interval* of $0.5sec$ and with 3 bins ($PLB = [0, 1250, 1500]$). The data is filtered, pre-processed and labelled appropriately per task before feeding it to ML models. For instance, for the application type classification task, we filter the top providers of each class and just associate the type as the final label. So, Video class, for example, has records from top providers (e.g. Netflix, Disney etc.) but just with the label "Video" after the pre-processing. Table 1 shows number of flows (approx.) that have been used for each task.

4.2. Vanilla DL Models

We now present a brief overview of CNN and LSTM models extensively used for NTC tasks and how FlowPrint can be fed into each model. DL models, as opposed to traditional ML models, are suitable since they automatically extract task-specific features from FlowPrint.

4.2.1. 1D CNN. CNNs are widely used in the domain of computer vision to perform tasks like image classification, object detection, segmentation and since recently are also being used in time-series classification tasks. Traditional CNNs (2-D CNNs) are inspired from visual circuitry in brains wherein a series of filters (also called as kernels) stride over an channeled (RGB) image along both height and width collecting patterns of interest for the task. However, 1-D CNN (where filters stride over 1 dimension of image) have been shown to be more effective for time-series classification objectives such as NTC tasks. Further, CNN's fast execution speed and spatial invariance makes them particularly suitable for NTC tasks [8].

FlowPrint needs no further processing to pass as an input to CNN (we omit 1-D for brevity) as it can be viewed as a colored image. Just as a regular image has height, width and 3 color channels (RGB), FlowPrint has bins (height), time slots (width) and, direction and counter types together forming six channels – upPackets,downPackets, upBytes, downBytes, upPacketLengths and downPacketLengths. Thus, FlowPrint is equivalent to a 6 channeled image of shape (numbins, timesteps, 6).

The CNN architecture (shown in Fig. 3-top) used in our work has 4 sub-modules each using a particular kernel size to perform multiple sequential convolutions on the FlowPrint image. The 4 kernel sizes used in our work are 3,5,7 and 9 along the timeslot axis i.e. their field of view includes all bins, all channels and timeslots equal to their kernel size. Using multiple sequential convolutions helps build features in a hierarchical way, summarizing to the most important features at the last convolutional layer. We use 8 layers as we found that results show marginal improvements on increasing the number of layers any further. The output from last layer of each module is flattened to a 32-dimensional vector using a dense layer, which is concatenated with the outputs of other modules. The concatenated output (32x4) is then passed to linear MLP (2 dense layers with 100 and 80 neurons) and then a softmax layer that outputs a probability distribution over the classes of the NTC task.

4.2.2. LSTM. LSTM is type of Recurrent neural network (RNN) widely used in tasks such as time series classification, sequence generation etc since they are designed to extract time-dependent features out of the raw input. LSTM processes a given sequence one time step at a time while remembering context from previous time steps by using hidden states and cell state that effectively mimic the concept of memory. After processing the entire input, it produces a condensed vector consisting of features extracted to perform the given task. Due to this, LSTMs have been used to perform NTC tasks [4], [9] in addition to CNNs.

Our FlowPrint arrays need to be reshaped to be fed into an LSTM model. We convert FlowPrint into a time-series vector $X = [X_0, X_1, X_2, ...X_T]$ where each X_t is a $3 * 2 * b$ dimensional vector consisting of values collected in time

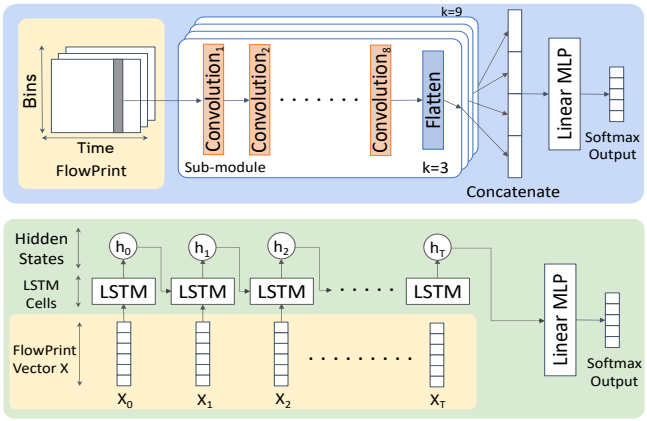

Figure 3: CNN (top) and LSTM (bottom) architectures.

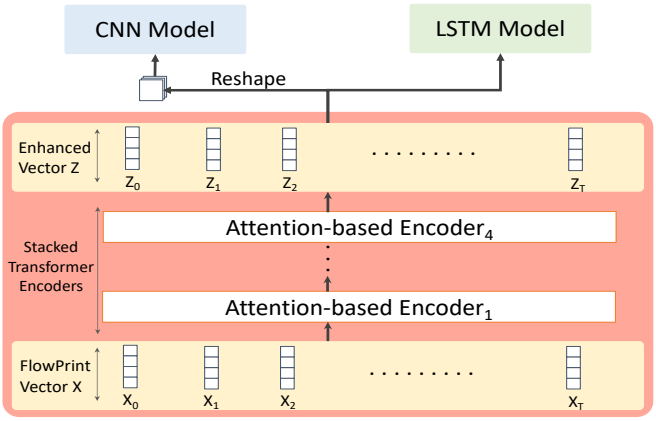

Figure 4: Transformer-based Architecture.

slot t, from 3 counters types (i.e. bytes, packets and packet lengths) collected in 2 directions (i.e up and down) and for b packet length bins i.e. all counters collected in time t.

The architecture used in our work (shown in Fig. 3 bottom) has one LSTM layer (of 80 neurons) which sequentially processes the input X while keeping a hidden state $h(t)$ and a cell state $c(t)$ (cell state omitted in the figure). At each time step t, the LSTM is fed X_t, and $h(t-1)$ and $c(t-1)$ from previous time steps, to produce new $h(t)$ and $c(t)$. The final hidden state $h(T)$ is then fed to a linear MLP and a softmax layer to generate a probability distribution over the classification labels.

4.3. FlowFormers

In order to improve the performance of CNN and LSTM based models on NTC tasks, we propose the use of Transformer Encoders on the input prior to feeding it into the vanilla model architectures. We first present a brief overview of Transformers, with a particular focus on its encoder which uses attention mechanism to enhance input features. We then develop two models by extending previous DL models using Transformer Encoders: TE-CNN and TE-LSTM respectively. We refer to TE-CNN and TE-LSTM as FlowFormers as they enhance **Flow**Print features using Trans**formers** to perform the NTC tasks.

Overview. Transformers [5] have become very popular in the field of NLP to perform tasks like text classification, text summarization, translation etc. A Transformer model has two parts an encoder and a decoder. The encoder extracts features from an input sequence and the decoder decodes according to the objective. For example, in task of German to English translation, the encoder will extract features from the German sentence and the decoder will decode them to generate the translated English sentence. For tasks like sentence classification only the feature extraction is required so the decoder part of the transformer is not used. Transformer encoder Models like BERT [15] are very effective in text classification tasks. Drawing inspiration from them, we develop a transformer encoder suited for NTC tasks.

Self-Attention. Transformer was able to outperform prior approaches in NLP due to one key innovation: Self-Attention. Prior to this, in NLP tasks, typically each word in a sentence was represented using a encoding vector independent of the context in which the word was used. For example, the word "Apple" was assigned same vector while it can refer to a fruit or the company depending on the context. A transformer encoder, on the other hand, uses a self attention mechanism in which other words in the sentence are considered to enhance the encoding of a particular word. For example while encoding this sentence, "As soon as the monkey sat on the branch it broke." Attention mechanism helps the transformer encoder to associate "it" with the branch, which is otherwise a non-trivial task.

Concretely, self attention works by assigning an importance score to all input vectors for each output vector. The encoder takes in a sequence $X_0, X_1, ...X_T$ where each X_t is a k dimensional input vector representing t-th word in the sentence. It outputs a sequence $Z_0, Z_1, ...Z_T$ where each Z_t is the enhanced encoding of the t-th word. For each Z_t, it learns the importance score c_t ($0 <= c_t <= 1$) to give to each input X_t, and then constructs Z_t as follows:

$$Z_t = \sum_{t=0}^{T} c_t.X_t, where \sum_{t=0}^{T} c_t = 1$$

This is just an intuitive overview of attention, the exact implementation details are described in [5].

Transformers for NTC. Similar to enhancing a word encoding, transformers can be used to enhance the time-series counters collected in FlowPrint. We implement this idea by developing an architecture for FlowFormers (shown in Fig. 4). FlowFormers TE-CNN and TE-LSTM are CNN and LSTM extended with Transformer Encoders. FlowPrint is first encoded by a Transformer Encoder before being fed into the CNN and LSTM architectures. In our work, we use 4 stacked transform encoders each with 6 attention heads. Each transformer encoder is designed exactly as in [5] with the dimensions of key, value and query set at 64.

The input format to the transformer encoder model is time-series vector X exactly similar to the input of LSTM (§4.2.2). The input is passed through multiple stacked encoders which enhance the input with attention at each level. We empirically found that using 4 stacked encoders gives

us the best results. The output of the final encoder is the enhanced vector Z exactly of the same dimensions as X. Now, instead of using raw FlowPrint X as input, we use enhanced version Z as an input to both models.

For TE-LSTM, the vector Z is directly fed into the LSTM model with no modification. For TE-CNN however, the vector Z is first converted into a 6-channel image (essentially, the reverse of the process of converting 6-channel image into input X described in §4.2.2). The image formatted input is then fed into the CNN model. We would like to highlight that since the input X and output Z are of exact same dimensions, the transformer encoder component is "pluggable" into the existing architectures requiring no modification to them.

Like most DL-models, the learning process even with transformer encoders is end-to-end i.e. all the model parameters including attention weights are learned by using stochastic gradient descent (SGD) and reducing the error of classification. Intuitively, in the case of TE-CNN, the CNN architecture updates the encoder weights to be more suitable to extract features using visual filters while in case of TE-LSTM, the LSTM updates the encoder weights to pick out time-series features. Irrespective of the vanilla model architecture used on top, transformer encoder is capable of enhancing the input such that it is amenable to how vanilla model works. This in turn makes the entire model (TE + vanilla model) learn and perform better compared to just the vanilla model architectures across the range of NTC tasks as shown in the evaluations next.

5. Training and Evaluation

Having explained the architectures of our DL-models, we now describe the training process followed by an evaluation of both FlowPrint and the models. As described in §4.1, we train our models for 2 tasks: (a) application type classification, (b) provider detection for video and conference application types. The dataset contains FlowPrint arrays labelled with both type and provider collected with the configuration mentioned in §4.1.

In addition to evaluating prediction performance of the models on the tasks above, we also evaluate the impact of parameters of FlowPrint. In particular, we evaluate the models' performance in various binning configurations and also with FlowPrint collected for a smaller duration i.e. 10sec and 20sec. For all these configurations, the training process remains the same as explained next.

5.1. Training

For each NTC task, the data is divided into train, validation and testing tests in the ratios 60%, 15% and 25% respectively. The data contains approximately equal examples from each class (for each task). All the DL models are trained for 15 epochs where in each epoch the entire dataset is fed to the model in batches of 64 at a time. Cross-entropy loss is calculated for each batch and then model parameters are learned through back-propagation using the standard Adam optimizer with an empirically tuned learning rate of 1e-4. After each epoch, the model is tested on the validation data and if the results on the validation data begin to drop, the training process is halted. This makes sure that the model is not over-fitting to the data it is being trained on, commonly known as early stopping in DL literature. These training parameters (and models' hyper-parameters) can be tuned specifically to perform slightly better. However, our aim in this work is to evaluate efficacies of different model architectures on FlowPrint as opposed to investigating specific tuning parameters for the models for each task. Hence, we keep the training process simple and consistent across all models and tasks to perform a fair comparison.

Table 2: Dataset split for type classification

App Type	Set-A Providers	Set-B Providers
Video	Netflix, Youtube, Disney	AmazonPrime, Facebook
Conferencing	MS teams, Zoom, Discord	Skype, Whatsapp , Hangout
Gameplay	Genshin Impact,LoL, CoD,WOW, CS:GO,	CoD: Black Ops Cold War, Fortnite, Overwatch, Halo Reach, Battlefront II, Hearthstone
Downloads	Steam , XboxLive	Playstation, Oculus, Origin
Live Video	Twitch , Seven Live	————————

5.2. Model Evaluation

We evaluate the vanilla models (CNN and LSTM) and FlowFormers (TE-CNN and TE-LSTM) on application type classification and provider classification tasks using FlowPrint input configured with 3 bins (0,1250,1500) and collected for 30 seconds (at 0.5 sec *interval* i.e. 60 time slots). We consider the commonly used metric f1-score (harmonic mean of precision and recall) as a measure of model performance across the tasks.

5.2.1. Type Classification. For application type classification task, we consider the following labels: Video, Live Video, Conferencing, Gameplay, Download. We divide the dataset into 2 mutually exclusive sets based on application providers: set A and B (as shown in Table 2). We train the model on 75% data (60% train and 15% validation) of set A, and perform two evaluations: 1) test on 25% of data in set A and 2) On all the data in set B. We note that the class "Live Video" has been excluded in this set as it contained only two providers.

The evaluation on set A (shown in Fig. 5-top), compares weighted and per-class f1 scores of both vanilla models (CNN, LSTM) and FlowFormers (TE-CNN and TE-LSTM). Firstly, all models have a weighted average f1-score of at least 92% indicating the effectiveness of FlowPrint to capture the traffic shape and distinguish application types. Secondly, FlowFormers consistently outperform vanilla models (by 2-6%) showing the impact of transformer encoders.

The evaluation on set B (shown in Fig. 5-bottom) tests the ability of models to learn provider-agnostic patterns to detect the application type since they were never shown examples from set B's providers. While the performance drops

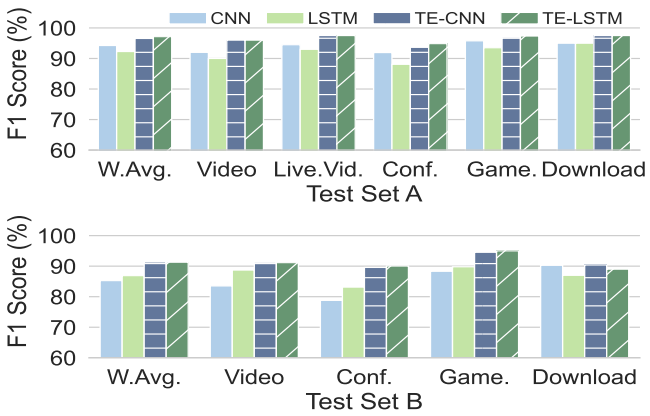

Figure 5: Type Classification.

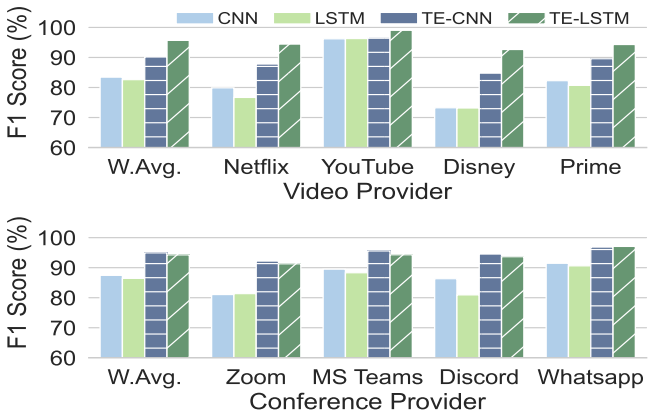

Figure 6: Provider Classification.

across models as expected, we observe that FlowFormers outperform vanilla models by a huge margin (6-11%). This clearly depicts that FlowFormers can generalize better than vanilla DL models due to attention-based encoders enhancing the FlowPrint input.

5.2.2. Provider Classification. For application provider classification, we aim to classify top providers amongst 2 application types: Video and Conferencing, i.e. classify amongst Netflix, YouTube, Disney and AmazonPrime for Video and Microsoft Teams, Zoom, Discord and WhatsApp for Conferencing. This task is inherently more challenging since all the providers belong to the same application type and hence largely have the same traffic shape. The models need to pick up on intricate patterns and dependencies such as packet length distribution and periodicity (in the case of video) to be able to classify amongst the providers.

For video provider classification (shown in Fig. 6-top), we observe that FlowFormers evidently perform better that the vanilla models with a 12% gain in the weighted average (e.g. TE-LSTM vs LSTM). We believe TE-LSTM outperforms other models since it can better pick up the periodic patterns (transfer of media followed by no activity shown in 2) that exist in the video applications. For instance, we observe (in our dataset) that YouTube transfers media every 2-5 seconds, whereas Netflix transfers it every 16 seconds. Transformers enrich FlowPrint by learning to augment this information and thus improving the classification accuracies.

Similarly in conference provider classification (shown in Fig. 6-bottom), FlowFormers outperform the vanilla models by 7% on an average (TE-CNN vs. CNN). We note that for this task, TE-CNN performs slightly better than TE-LSTM since this task predominantly relies on packet length distributions which tend to be different for the providers of conferencing applications rather than periodic patterns observed in video applications.

To summarize, FlowFormers are able to learn complex patterns beyond just the traffic shape, to outperform vanilla models in the challenging tasks of video provider and conference provider classification.

5.3. FlowPrint Evaluation

We now evaluate the performance of FlowPrint by varying number of bins and length of data. We show that FlowPrint's binning is a key factor that increases the performance across models, especially in the challenging task of provider classification.

5.3.1. Bin analysis. In previous evaluations, each FlowPrint sample had 3 bins ($PLB = [0, 1250, 1500]$). Now, we evaluate the impact of reducing bins to 2 ($PLB = [1250, 1500]$) and 1 ($PLB = [1500]$) on the performance of the models. We note that to reduce to 2 bins we have two choices, either (a) merge bin 2 and bin 3 or (b) merge bin 1 and bin 2 in the original 3 bin configuration. We chose to merge bin 1 and bin 2, since that was giving a better performance. So, in other words, the 2-bin configuration tracks the counters in less-than-MTU ($0 <= pkt.len <= 1250$) and close-to-MTU bins ($>= 1250$). We additionally note that 1 bin, essentially means that there is no binning at all i.e. FlowPrint with 1 bin tracks the total byte and packet counts of the flow without any packet length based separation.

We re-train and evaluate every model on each of the 3 bin configurations for the tasks Application Type Classification and Video Provider Classification (weighted average f1 scores shown in Fig. 7). We observe that the f1 scores across the models and tasks generally improve with more bins. However, the performance improvement also depends on the task complexity. For type classification task, the models improve by less than 2% per addition of a band (the difference is further insignificant for FlowFormers). For video provider classification however, the performance increment is evidently drastic since the task is more challenging and requires fine grained data with binning. On the other hand, Conference Provider Classification (not shown in the figure), has little to no impact on f1 scores by reducing bands as almost all packets exchanged within the one bin ($0 <= pkt.len <= 1250$).

Thus, the configuration of FlowPrint can be decided depending upon the NTC task at hand. Higher number of bins would imply higher memory footprint which is especially expensive in programmable switches which have very

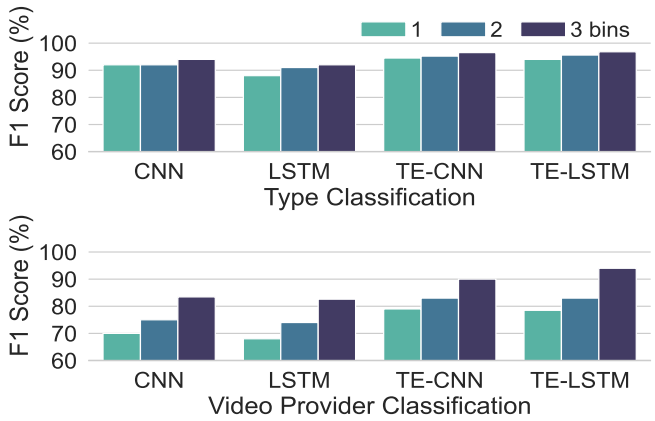

Figure 7: FlowPrint Bin Results.

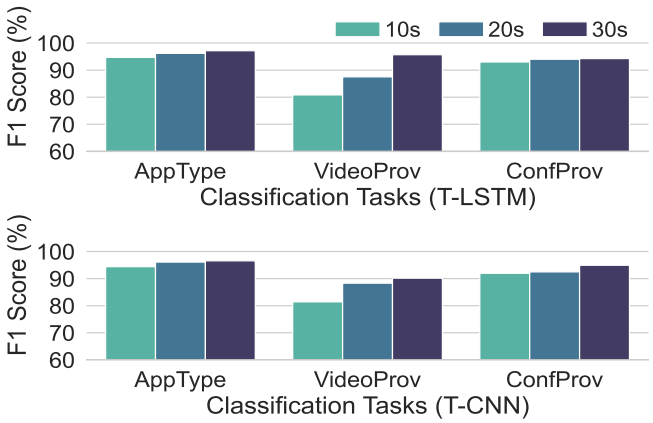

Figure 8: FlowPrint Time Results.

limited memory. So, this evaluation helps to navigate the bin vs. memory tradeoff to configure FlowPrint parameters and achieve a particular target accuracy for an NTC task.

5.3.2. Time Period Analysis. We now evaluate the impact of the time period for which FlowPrint is collected on each task. We re-train and evaluate FlowFormers (vanilla models omitted for brevity) on FlowPrint collected for 10sec, 20sec and 30sec (the max configuration). The Fig. 8 shows the weighted average f1-score of TE-LSTM (top) and TE-CNN across the tasks (x-axis). We note that both models are able to classify application types with about 95 % f1 score with just 10 seconds of data while going up to 97% with 30 seconds. Similarly, the conferencing provider classification results do not vary by much with increasing time as a conference call tends to exhibit similar behaviour over the given time range. However, for video provider classification task, we can observe a significant gain by using FlowPrint collected for a longer duration. This is due to the periodic nature of the flows which repeats at a longer interval (e.g. 16 seconds for Netflix).

Thus, the parameters of FlowPrint i.e. *numbins*, *time duration*, *interval* etc. can be configured depending upon the NTC task, the available compute/memory resources and required performance in terms of classification speed and overall accuracy.

6. Conclusion

With diverse applications being used on the internet, *Network Traffic Classification* is becoming important but increasingly challenging due to encryption. Existing approaches either rely on non-encrypted content of traffic or perform post-facto classification of flows. Our work has developed methods for accurately classifying traffic in real-time and at scale by using only the behavioural patterns agnostic to flow content. To this end, we design *FlowPrint*, a data-structure to efficiently capture traffic behaviour that is amenable to implementation in high-speed programmable switches. We further propose the use of transformer-encoders (*FlowFormers*) to outperform existing DL models. Our evaluations show that the combination of *FlowPrint* and *FlowFormers* can classify application type and providers at scale with high accuracies and in real-time.

References

[1] Chrome Team, "A safer default for navigation: HTTPS," https://bit.ly/2V1KWrS, 2021.

[2] S. Rezaei *et al.*, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2019.

[3] H. H. o. Gharakheili, "itelescope: Softwarized network middle-box for real-time video telemetry and classification," *IEEE TNSM*, vol. 16, no. 3, pp. 1071–1085, 2019.

[4] 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. [Online]. Available: https://doi.org/10.1145/3447382

[5] A. Vaswani *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[6] M. Lotfollahi *et al.*, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[7] P. Wang *et al.*, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55 380–55 391, 2018.

[8] W. Wang *et al.*, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 Proc. IEEE ISI*, 2017, pp. 43–48.

[9] G. Aceto *et al.*, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE TNSM*, vol. 16, no. 2, pp. 445–458, 2019.

[10] G. Draper-Gil *et al.*, "Characterization of encrypted and vpn traffic using time-related," in *Proc. ICISSP)*, 2016, pp. 407–414.

[11] M. Lopez-Martin *et al.*, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[12] Z. Chen *et al.*, "Length matters: Fast internet encrypted traffic service classification based on multi-pdu lengths," in *2020 16th IEEE MSN*, 2020, pp. 531–538.

[13] V. Rimmer *et al.*, "Automated website fingerprinting through deep learning," *arXiv preprint arXiv:1708.06376*, 2017.

[14] Intel, "Tofino," https://intel.ly/3y0EIHg, 2021.

[15] J. Devlin *et al.*, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.