Heavy-Hitter Flow Identification in Data Centre Networks Using Packet Size Distribution and Template Matching

Alejandra Duque-Torres*, Adrian Pekar†, Winston K.G. Seah‡, and Oscar Mauricio Caicedo Rendon*
*Department of Telematics Engineering, University of Cauca, Popayan, Colombia
E-mail: {aduquet, omcaicedo}@unicauca.edu.co
†Department of Networked Systems and Services, Budapest University of Technology and Economics, Hungary
E-mail: apekar@hit.bme.hu
‡School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
E-mail: winston.seah@ecs.vuw.ac.nz

***Abstract*—Data Centre Networks (DCNs) handle large volumes of data transmission that can consume a lot of bandwidth in short bursts or over prolonged periods of time. One class of traffic that constantly poses a challenge is Heavy-Hitter (HH) flows – large-volume flows that consume considerably more network resources than other flows combined. The identification of such flows is critical to prevent network congestion and overall network performance degradation. Most of the existing methods to identify HHs are based on thresholds, i.e., if the flow exceeds a predefined threshold, it will be marked as a HH; otherwise, it will be classified as a non-HH. However, these approaches present two significant issues. First, there is no consistent and accepted threshold that would reliably classify flows. Second, the existing threshold approaches use counters (duration, packets, and bytes); thus their accuracy depends on how complete the flow information is. In this paper, we address those issues using per-flow packet size distribution which can capture the behaviour and dynamics of network traffic flow more accurately than the counters in the early stage of the flow. We then propose the use of the template matching technique to identify HHs and achieved a classification accuracy of 96% using only the first 14 packets of a flow.**

***Index Terms*—heavy-hitters, flow, elephant, mice, data centre networks, packet size distribution, threshold, software-defined networking, template matching**

I. Introduction

In the last few decades, DCNs have become vital components of a wide range of applications, such as big data processing, cloud computing infrastructure, and multimedia content delivery. A traditional DCN comprises a very high number of network devices that support the seamless exchange of traffic between the virtual machines/servers and the Internet. These devices include switches that interconnect hosts, routers that forward the traffic, and gateways that act as a junction between the DCN and the Internet. The main DCN limitations have been scalability and growing management complexity.

To overcome these limitations, new networking paradigms such as Software-Defined Networking (SDN) have been introduced into DCNs. SDN is an emerging technology, in which the control plane is decoupled from the forwarding plane. By moving the control logic from the forwarding devices to a logically centralised device, namely, the controller, the network devices become simple forwarders that are programmable through a standardised protocol such as OpenFlow [1]. Data centres implemented using SDN are referred to as Software-Defined Data Centre Networks (SDDCNs).

One of the main benefits of SDDCN is the centralised view of the network and, most importantly, its traffic flows. However, the clear and well-defined benefits of central network traffic control cannot guarantee that the network performance will not degrade when traffic volume increases. Applications and services have increasing Quality of Service (QoS) requirements. In consequence, managing the traffic that flows through the network remains a challenge [2]. An essential tool to ensure the reliable operation of networks is traffic flow classification, which provides inputs for a variety of network managerial related activities [3]. In network performance maintenance a significant objective is to identify and classify flows that exhibit heavy-tailed or long-tailed distribution.

Examination of heavy-tailed distribution in flows has been an objective of several research efforts. An inferable observation, that can be deduced from these works is that a very small percentage of flows carry the bulk of the traffic [4], [5]. These flows are most commonly referred to as Heavy-Hitter (HH) or elephant flows. One of the consequences of unsupervised forwarding of HH flows is, among others, that it often leads to network congestion and, subsequently, to overall network performance degradation [4], [6]. The identification of HHs provides inputs for a variety of network managerial related activities, such as flow scheduling, QoS provisioning, and load balancing.

In this paper, we propose a novel HH identification approach based on packet size distribution (PSD) and template matching (TM). Our evaluation shows that our approach achieves up to 96% accuracy while using only the first 14 packets of a flow. Furthermore, this accuracy remains consistent throughout all classifications while existing approaches yield different accuracies for different flow size-based thresholds.

TABLE I: Related work in the domain of SDDCN

Work	Citations	Year	Appeared In	Threshold	Value	Static	Adaptive
Curtis *et al.* [7]	326	2011	IEEE INFOCOM	Flow Size	128KB, 1MB and 100MB	✓	
Sivaraman *et al.* [8]	79	2017	Symposium on SDN Research	Number of packets	Not specified		✓
Chiesa *et al.* [9]	68	2016	IEEE/ACM Trans. Netw.	Bandwidth	10%	✓	
Trestian *et al.* [10]	32	2013	IFIP/IEEE IM	Rate	Not specified	✓	
Liu *et al.* [11]	24	2013	IEEE Commun. Lett.	Rate	1-10 Mbps	✓	
Lin *et al.* [12]	18	2014	IEEE GLOBECOM	Flow Size	100 MB	✓	
Bi *et al.* [13]	11	2013	IEEE GLOBECOM	Flow Size	Not specified		✓
Poupart *et al.* [14]	10	2016	IEEE ICNP Workshop	Flow Size	10 Kb to 1 Mb	✓	
Liu *et al.* [15]	2	2017	Wiley Int. J. Netw. Man.	Flow Size	Not specified		✓

The remainder of this paper is organised as follows. In Section II, we review related work while Section III discusses PSD and its efficiency in HH detection. Then, in Section IV, we present the details of our HH classification technique. Section V introduces the measurement outcomes and discusses the achieved results. Finally, we conclude this work in Section VI.

II. Related Work

In SDDCNs, the SDN controller has central awareness and control of the flows. The main role of an SDN controller is to instruct the devices how the network should accommodate the forwarding of the traffic. The main features used to achieve this are flow counters, specifically *duration*, *packet count*, and *bytes*. Assuming that we are able to achieve a central view of all flows and reliable collection of flow counters' data, then performing HH detection is simple: first, we set a threshold and then all flows exceeding this threshold are classified as HHs while flows below this threshold are classified as non-HHs. Despite this simple methodology, HH identification is surrounded by a number of challenges from data collection (*i.e.,* achieving the central view of all flows) to threshold determination and classification [7], [9], [11].

In this work, we focus on the classification aspect of HH-detection. Table I summarises related works in the domain of SDDCN. A common observation from Table I is that there is no consensus about the threshold that would reliably separate the flows into HHs and non-HHs, nor regarding the feature or set of features (*e.g.,* size, duration, and rate) that should be used for HH identification. In general, two HH classification approaches exist: *static* and *dynamic* threshold-based approaches.

In the former, the threshold is usually determined at the beginning of the detection phase and this threshold remains unchanged until the end of the measurement or unless the detector is required to be reconfigured and restarted. While a static threshold HH detection mechanism is relatively simple to implement, this approach is considered to be inaccurate due to its inability to adapt to changes in the traffic behaviour. In the latter, also often referred to as the adaptive approach, the threshold is adjusted in specific time intervals based on changes in conditions such as the variation of the traffic load.

III. Per-Flow Packet Size Distribution

A flow is defined as a set of packets passing an observation point during a specific duration of time [16]. All the packets that belong to the same flow share a number of attributes, specifically, the source and destination IP addresses, source and destination port numbers and the protocol identifier number. To identify a flow as HH, a certain number of packets must be observed before a decision can be made. This number is usually kept at a minimum to achieve real-time classification. However, the packets belonging to various flows pass through the data plane in an indeterministic sequence. As a result, it is not uncommon to see that packets belonging to the same flow are captured with intervals of more than tens of seconds. Therefore, to achieve real-time classification, a time limit for per-flow packet collection is usually implemented. This way the identifier does not have to wait too long to see a specific number of packets before it makes a decision. However, the classification accuracy is challenging when at any given period only part of the flow is known or if there are too few packets in the flow.

To overcome this challenge, a feature is required that captures the flow behaviour (HH or non-HH) well even if only a few packets are known in the flow. We argue that per-flow PSD captures the behaviour and dynamics of network traffic flows as accurately as the traditional counters (duration, packets, and bytes) that are used by existing HH identification approaches, especially with respect to their heavy-tailed distribution. The advantage of PSD over traditional counters is that it can perform reliably even if only limited information is available about the flow.

An important consideration when analysing the viability of PSD for identifying HH flows is the minimum number of packets. This number represents a 'window' in which the PSDs are going to be analysed. To determine the size

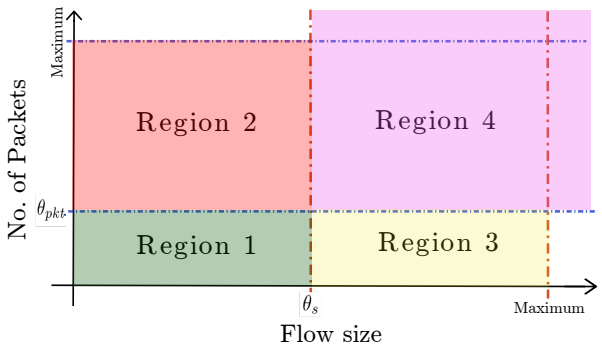
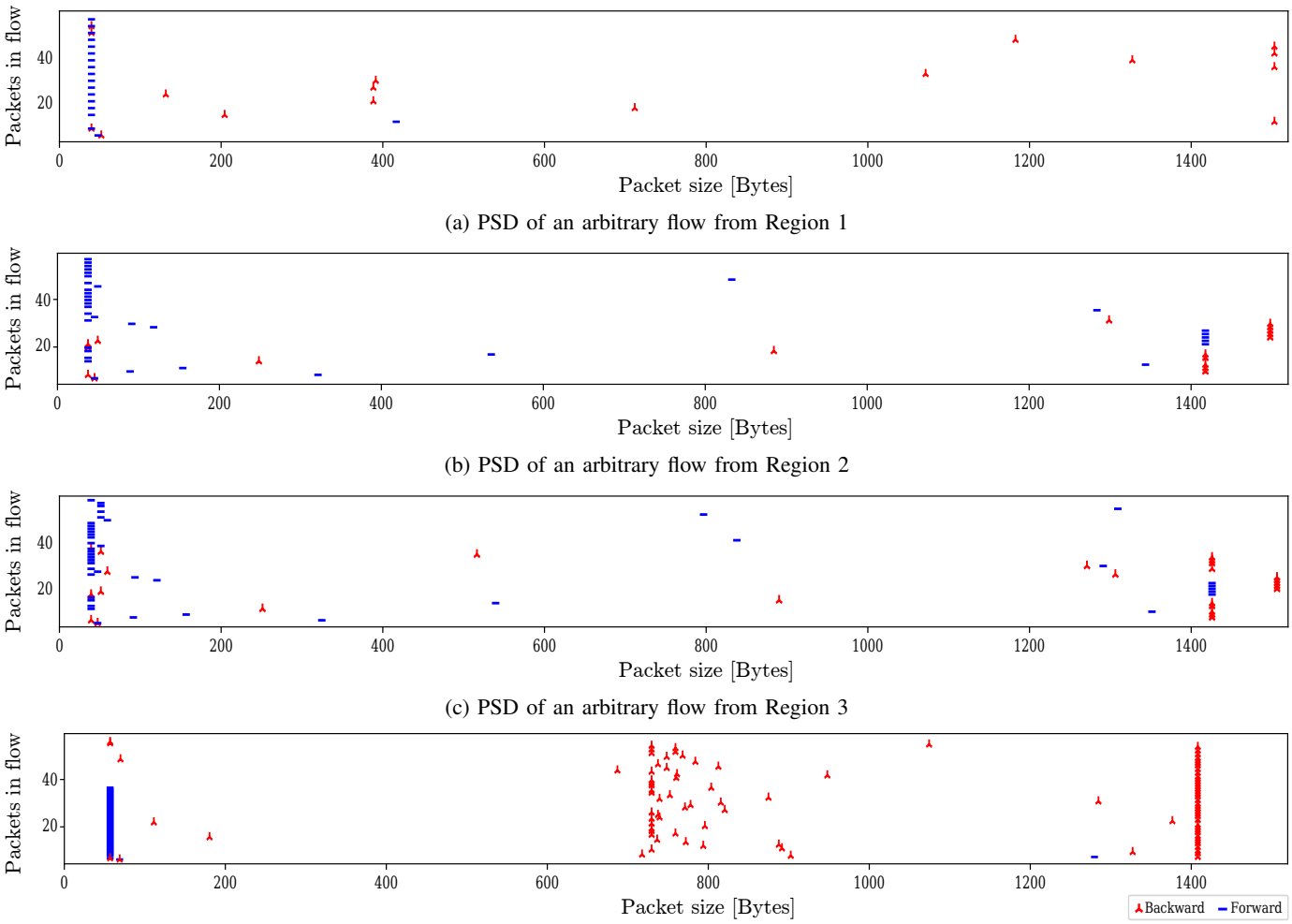


Fig. 1: Regions created by θ_s and θ_{pkt}



(a) PSD of an arbitrary flow from Region 1

(b) PSD of an arbitrary flow from Region 2

(c) PSD of an arbitrary flow from Region 3

(d) PSD of an arbitrary flow from Region 4

Fig. 2: Flow samples with various per-flow PSDs

of this 'window', we performed clustering-based analysis on the UNIV1 [4] dataset as a test case. The results of this clustering-based analysis are available at [17]. We examined the unlabelled flows and observed that they can be classified into four classes (clusters). The threshold of these classes can be clearly defined using the flow size θ_s and the packet count θ_{pkt} features. In UNIV1, the optimal threshold values are $\theta_s = 6$ KB and $\theta_{pkt} = 14$ [17]. These two thresholds classify each flow in UNIV1 into exactly one of the four regions as depicted in Figure 1.

Figure 2 provides an example of the packet size distribution of an arbitrary flow belonging to each of the four regions. The flows in Regions 1 (Figure 2(a)) and 4 (Figure 2(d)) represent the minimums and maximums, *i.e.,* non-HHs and HHs, respectively. The difference in terms of per-flow PSD is clear and simple to recognise without the need for a detailed investigation of the flow. Regions 2 (Figure 2(b)) and 3 (Figure 2(c)) are, however, not that obvious as the flows exhibit similar characteristics in terms of PSD. The classification of these flows requires an in-depth investigation.

IV. System Design

In general, per-flow PSD can be considered as a collection of points (see Figure 2). To obtain more efficient representation of PSD, we can also consider these packet sizes as a continuous random variable with different values. Then, we can generate a function which describes how such variable is distributed in a range given. The *probability density function* (*pdf*) is a statistical feature that is suitable for describing this phenomenon. It defines the probability distribution of a continuous random variable.

We denote a continuous random variable representing each packet size belonging to a certain flow as x_{pkt}. The *pdf* provides the value of the function at any given x_{pkt}. We emphasise that *pdf* does not directly provide the probability of x_{pkt}. Instead, it yields the probability P that x_{pkt} will take on a value within a given interval $[a, b]$. This can be formally expressed as:

$$P[a <= x_{pkt} <= b] = \int_a^b pdf(x_{pkt})dx_{pkt}. \quad (1)$$

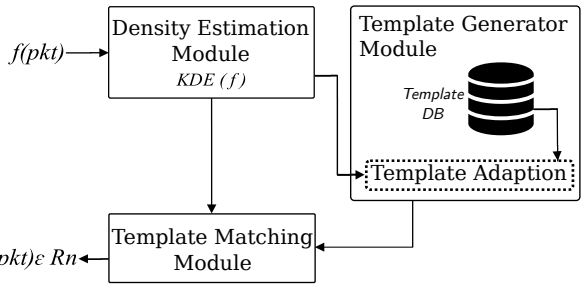


Fig. 3: Architecture of the system prototype

Our approach to identify HH is based on per-flow PSD and TM. The PSD is able to capture the behaviour and dynamics of network traffic, and TM [18] is used for pattern recognition.

A crucial point of TM when used in pattern recognition is to adopt an appropriate "measure" to quantify similarity given an *input* and a *template*. In our case, the *input* is a flow, which is expressed as

$$f_s = \sum_{i=1}^{N} x_{pkt}(i), \tag{2}$$

where x_{pkt} represents the size of packet belonging to a certain flow. The *templates* represent the flow size behaviours in a particular region, i.e. a *pdf* class. Our system provides a similarity measure per-template given an input. The higher the similarity, the more probable is that the input (*i.e.,* a flow) belongs to the region.

Figure 3 shows the architecture of our approach. It is composed of three main components: *Density Estimation*, *Template Generator*, and *Template Matching*.

A. Density Estimation Module

This module is responsible for estimating the *pdf* of the incoming flow. We can consider the flow packet sizes as random variables. Several techniques exist to estimate the *pdf* of a random variable. In general, the density estimator can be classified into two types: *parametric* and *non-parametric* [19]. The parametric estimators need a fixed form or structure of the data and depend on the previous data point while the non-parametric estimators have no fixed structure and depend on all the input data points. We use Kernel Density Estimation (kde) to estimate $pdf(x_{pkt})$; formally expressed as:

$$kde(x_{pkt}) = \frac{1}{N} \sum_{i=1}^{N} K_h(x - x_i), \tag{3}$$

where K_h is a kernel function, and x represents each point of x_{pkt}. The kde is a non-parametric method in which does not require any preceding models. Moreover, in contrast to other non-parametric methods such as the histograms, its density estimates are smoother, continuous and differentiable [20].

We use *Gaussian Kernel* to estimate the *pdf* since it has been shown to yield good performance under general smoothness assumptions. In addition, it has good performance when no additional knowledge of the data is available [21].

B. Template Generator

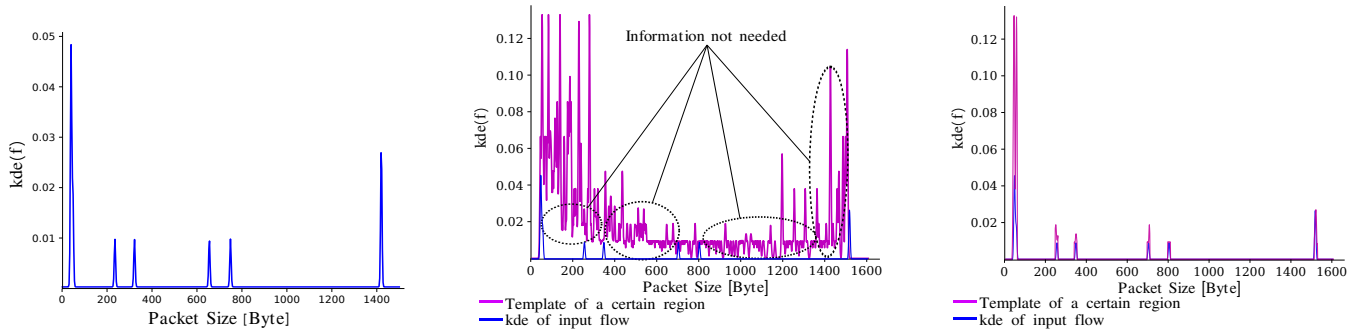
This module is composed of the *Template Database* and *Template Adaption* components. *Template Database* is responsible for storing the individual *templates*. Consider $R = [R_1,\ R_2,\ R_3,\ R_4]$ as the set of the four regions introduced in Figure 2 while $R_n = [f_{s_1},\ f_{s_2},\ ...,\ f_{s_N}]$ denotes an arbitrary region composed of a set of N flows.

Templates per each region are generated based on the TemplateGenerator procedure as per Algorithm 1. This procedure has three main functions: pktExtractor, kdeEstimation, and FinalTemplate. pktExtractor is responsible for extracting the first θ_{pkt} packets that are necessary to be collected before a flow can be classified. pdfEstimation computes the *pdf* estimation. The output of this function is a template for each flow in R_n. In our prototype implementation in Python, each template is a floating point real value while these values (templates) are stored (per each region) as a list of floats.

Obviously, the number of templates in R_n depends on the numbers of flows in the region. This, however, can lead to challenges in real-world applications (*e.g.,* in terms of resource, time, and memory constraints). To minimise the computing demands, we reduce the number of templates per each region. More specifically, instead of maintaining hundreds of thousands of templates, we compute only one

Algorithm 1 TemplateGenerator

procedure TEMPLATEGENERATOR(i, θ_{pkt})
  ▷ *pd*: Python *pandas* library
  function FINALTEMPLATE(X_{kde}, Y_{kde})
    global mainT
    if len(mainT) ! = 0 **then**
      for j **in** range(0, len(Y_{kde})): **do**
        y_{aux} = mainT[y][i].$max()$
        $y_{aux_2} = Y_{kde}$[i].$max()$
        if $y_{aux} < y_{aux_2}$ **then**
          mainT = {'x': X_{kde},'y': y_{aux_2}}
        else
          mainT = {'x': X_{kde},'y': y_{aux}}
    else
      mainT = {'x': X_{kde},'y': Y_{kde}}
    return $Template$
  function PDFESTIMATION(y)
    kde = KernelDensity(kernel, bandwidth)
    kde.fit(Y_{kde}[:, None])
    return X_{kde}, Y_{kde}
  function PKTEXTRACTOR(i, θ_{pkt})
    for index, row **in** i.iterrows: **do**
      if index < len(i)−1 **then**
        flowIDaux = i[index+1, f_{key}]
        if flowIDaux ! = i[index, f_{key}] **then**
          data = $i.copy()$
          data.$drop$[data[f_{key}] != i[index−1, f_{key}]
          y = data.$head(\theta_{pkt})$
          if len(y) == θ_{pkt} **then**
            X_{kde}, Y_{kde} = KDEESTIMATION(y)
            r = FINALTEMPLATE(X_{kde}, Y_{kde})
    return r ▷ Final template
  Template = PKTEXTRACTOR(i, θ_{pkt})



(a) Example of a *pdf* of a flow to be classified that is calculated by the *Density Estimation Module*

(b) Matching the *pdf* of an arbitrary flow and a template that is computationally complex due to the information that is not necessary to achieve accurate classification

(c) Adapting the template to improve performance by preserving only those the data points that are required for accurate classification

Fig. 4: Template adaptation for efficient matching

'master' template per each region. To achieve this, we use the FinalTemplate function that computes the maximum value of each template in a particular region. Obviously, as new flows are observed, new templates are generated and so the maximum values continuously update the final template.

Lastly, *Template Adaption* adapts the templates to the *pdf* of the observed flow (that is going to be classified). Consider Figure 4(b) as the process that compares a *pdf* of a flow (see Figure 4(a)) calculated by the *Density Estimation Module* to a template of a particular region provided by the *Template Database*. From Figure 4(b) is evident that not all the data points are equally significant in terms of matching. In addition, the point-by-point comparison of each data point between the flow to be classified and the template is a computationally complex task especially with respect to memory and computing resources. To improve the performance of our solution, before the matching takes place, we adapt the template to the input flow so that only those data points that are required for accurate classification are preserved. An example of such adaptation is provided in Figure 4(c).

C. Template Matching Module

This module is responsible for performing a similarity measure between the *pdf* of the observed flows and the templates. In practice, this means that the *pdf* is compared with each 'master' template (one out of the four) in the template database and is classified into that class that yields the highest similarity measure. The similarity measure is usually calculated using distance or correlation metrics [18]. In our work, we specifically use the *Pearson correlation* and *Dynamic Time Warp* (DTW) similarity measures.

Pearson correlation is a number between -1 and 1 that indicates how much two variables are linearly related. The computational demand of obtaining Pearson correlation is relatively low while sensitive to a linear relationship between the variables [22]. A correlation of -1 indicates that the input and the observed template have a perfect inverse linear correlation. A correlation of 0 means no linear relation. Finally, a correlation equal to 1 means that both the input and the template have a perfect direct linear correlation. DTW, on the other hand, is a distance measure which allows two-time series that are similar but locally out of phase to align in a non-linear manner [23]. DTW computation follows three major steps [23]:

1) Compare each point in $pdf(f)$ with every point in X_r, generating a matrix.
2) Work through the matrix and calculate the lowest cumulative distance for each cell. Subsequently, add the value to the distance of the focal cell.
3) The distance between the two signals is then calculated based on the most efficient pathway through the matrix.

Similarity measure between the input and each template is implemented based on Algorithm 2. We query the templates stored in the *Template Database*, and perform similarity measurement first using Pearson correlation and then with DTW. The Pearson correlation is computed using the Python `scipy` library [24] while DTW is calculated via the `DTAIDistance` library [25]. As we will demonstrate in Section V, the combination of Pearson correlation and DTW improves the results of TM.

V. Results

We evaluate our approach using the UNIV1 [4] traffic trace that was collected in a university DCN. We processed and

Algorithm 2 Match

1: **procedure** MATCH(f_{pkt}, T_n)
2: ▷ f_{pkt}: *pdf* of incoming flow
3: ▷ T_n: Template per-region
4: $T_n = [\ X_{r_1}, X_{r_2}, X_{r_3}, X_{r_4}\ ]$
5: corrResult $[1] =$ pearsonr(X_{r_1}, f_{pkt}) ▷ Region 1
6: DWTResults $[1] =$ dtw.warping(X_{r_1}, f_{pkt})
7: corrResult $[2] =$ pearsonr(X_{r_2}, f_{pkt}) ▷ Region 2
8: DWTResults $[2] =$ dtw.warping(X_{r_2}, f_{pkt})
9: corrResult $[3] =$ pearsonr(X_{r_3}, f_{pkt}) ▷ Region 3
10: DWTResults $[3] =$ dtw.warping(X_{r_3}, f_{pkt})
11: corrResult $[4] =$ pearsonr(X_{r_4}, f_{pkt}) ▷ Region 4
12: DWTResults $[4] =$ dtw.warping(X_{r_4}, f_{pkt})

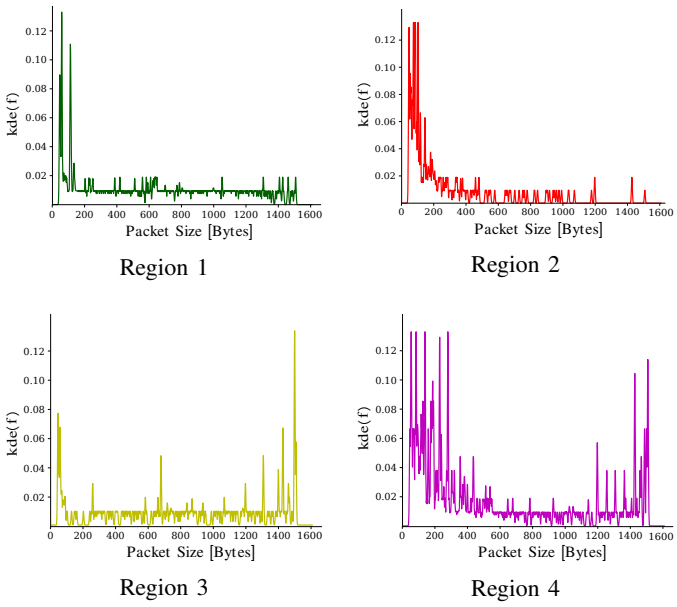


Fig. 5: Template per-region for UNIV1

organised UNIV1 into flow records with an expiration time set to 150 s. The obtained data set contains 34 788 flows out of which 13 457 flows belong to R_1, 6894 to R_2, 391 to R_3, and 14 046 to R_4. This dataset was split into training and test sets with a $70/30$ ratio while cross validation was also used to increase the effectiveness of the model. We used the training set to compute the templates.

Following the six-step knowledge discovery process [26], we determined the minimum number of packets before classification is performed. Our Silhouette analysis of the dataset returned $k = 4$ as the optimal number for clustering. Subsequently, using K-Means, we obtained the average number of packet counts, specifically 14. For a more detailed treatment of the clustering, the interested reader may consult [17]. To create the templates for the regions, as shown Figure 1, besides $\theta_{pkt} = 14$ we also used a further threshold that appeared to be the optimal number of minimum size (bytes) of flows, i.e. $\theta_s = 6$ KB. This value was also determined via clustering-based analysis [17].

A. Performance evaluation

The performance measures in this paper are given by five metrics derived from the *Confusion Matrix*, viz., *True Positive Rate* (TPR), *False Positive Rate* (FPR), *Accuracy*, *Precision*, and *f-measure*. Table III shows a typical confusion matrix of a binary classification. The *True Positive* (TP) represents a successful classification, e.g., a particular sample belonging to R_n, which was classified in class R_n. *True Negative* (TN) depicts successful classification in which the sample did not belong to R_n and was classified as Non-R_n. *False Positive* (FP) refers to an unsuccessful classification, *e.g.,* a sample belonging to Non-R_n was classified as R_n. *False Negative* (FN) represents an unsuccessful classification as well. In this case, a sample belongs to R_n is classified as Non-R_n.

TABLE III: Confusion Matrix for R_n

A' \ A	R_n	Non-R_n
R_n	TP	FP
Non-R_n	FN	TN

The classification system has been trained to distinguish between R_1, R_2, R_3, and R_4. We conduct the performance measures for each region. Table IV reports the confusion matrix for each region. Supplementary to Table IV, the performance measures are summarised in Table V.

The results obtained indicate that the region with highest TPR is Region 3, with TPR value of 0.9847, and the reason behind this is related to the number of flows belonging to this region. In contrast to the other regions, Region 3 contains very few flows, roughly 391. It is likely that, when the templates were generated, most of the flows belonged to Region 3. This results in a better representation of the flow behaviour. The region having the lowest TPR 0.8860 is the Region 2. This can be regarded as the opposite of Region 3, *i.e.*, when the templates were generated, few flows belonged to Region 2. Overall, our approach achieves a TPR of 0.9385 which means our approach has high sensitivity.

Our best FPR value is 0.002 which corresponds to Region 4. The highest value obtained is for Region 1 with 0.037. Notwithstanding this region has the highest value, it still is near to zero, which indicates good performance. The general result obtained shows an average of 0.020 which means that our approach has a low false positive rate.

The best accuracy result is 1, whereas the worst is 0. Region 3 has the accuracy of 0.9734, while Region 2 is the region with lowest value, 0.9582. The reason is the same as that for TPR. Our approach achieves an average accuracy of 0.96.

There is a natural trade-off between TPR and precision, which is why the region that performs best on one measure is not usually the one that perform best on the other measure. This is the case of Region 3. The precision value for Region 3 is 0.2952, while the other regions' values are over 0.9018. Such a low value for the Region 3 indicates that flows belonging to the other regions tend to be classified as Region 3 mostly. The overall precision is 0.7841 giving our approach an acceptable false positive rate.

An *f-measure* score reaches its best value at 1 and worst value at 0. A low *f-measure* score is an indication of both poor precision and poor recall. In our case the average of both recall and precision is high. Thus, the intuitive *f-measure* score for our approach is high, nearly 1.

Although we have achieved promising results in our experiments, we did not examine the impact of time. Time is a critical factor in HH detection. The packets belonging to various flows pass through the data plane in an indeterministic sequence. As a result, it is not uncommon for packets belonging to the same flow to be captured with intervals of more than tens of seconds. In real-time HH detection, waiting for tens of seconds or longer is not acceptable. For instance, we can achieve an accuracy of up to 96%, when classifying

TABLE IV: Confusion matrix per-region

Region 1

A' \ A	R_1	Non-R_1
R_1	12 930	780
Non-R_1	527	20 080

Region 2

A' \ A	R_2	Non-R_2
R_2	6108	667
Non-R_2	786	27 227

Region 3

A' \ A	R_3	Non-R_3
R_3	385	919
Non-R_3	6	33 478

Region 4

A' \ A	R_4	Non-R_4
R_4	12 955	44
Non-R_4	1091	20 698

TABLE V: Performance measures for UNIV1 dataset

	TPR	**FPR**	**Accuracy**	**Precision**	**f-measure**
$\boldsymbol{R_1}$	0.9608	0.037	0.9619	0.9431	0.9519
$\boldsymbol{R_2}$	0.8860	0.024	0.9582	0.9018	0.8937
$\boldsymbol{R_3}$	0.9847	0.027	0.9734	0.2952	0.9628
$\boldsymbol{R_4}$	0.9223	0.002	0.9673	0.9966	0.9580
Mean	0.9385	0.020	0.9623	0.7841	0.9362

UNIV1 with $\theta_s = 6$, 10, 100 and 1000 KB and $\theta_{pkt} = 14$ packets. However, for some flows, it takes up to 460 seconds to collect enough packets/bytes to meet these thresholds, which is unacceptable for timely classification. A time limit set for per-flow packet collection could resolve this problem simply. This way the classifier does not have to wait too long to see a specific number of packets before it makes a decision. However, this also raises degradation in accuracy due to too few packets captured per flow within the time duration. Real-time classification might be achieved but at the expense of collecting too few packets to make an accurate detection. As such, in future work we plan to perform a rigorous examination of our approach especially with respect to the impact of time.

B. Performance comparison

We compared our result with classification techniques used by Poupart *et al.* [14]. They report only the TPR obtained using *Neural Networks* (NN), *Gaussian Processes Regression* (GPR), and *Online Bayesian Moment Matching* (oBMM) to classify HHs. NN is one of the most flexible predictors in the sense that it can approximate any function when using sufficiently many nodes and data. The GPR models are non-parametric kernel-based probabilistic models. oBMM is a tractable online Bayesian learning algorithm for learning mixture models using the method of moments [14]. We made sure to adopt the same thresholds applied by Poupart *et al.* [14].

Figure 6 shows the TPR of each algorithm, and our approach which is denoted as TM. It is evident that our approach achieves only comparable results. However, there are some significant considerations. First, we compare only TPR results as Poupart *et al.* [14] does not provide any information on the other measures that we also calculated. As such, the comparison is only partial while it is also likely that the thresholds selected by Poupart *et al.* were selected optimally for their purposes. Second, GPR and NN cannot maintain the same performance when the threshold is changed. Our TM approach achieves the same performance in different thresholds. Third, the NN and oBMM approaches tend to be affected by class imbalances more than the Gaussian process, which explains why their accuracy often suffers as the classification threshold increases. In our approach, the imbalance in the regions is not a problem, since each flow is analysed in the same packet window. Also, the template is generated by the same packet window. Lastly, considering Table V, the achieved results are promising. Per-flow PSD is capable of capturing the behaviour and dynamics of traffic flows and as such, it is suitable for HH detection. Using TM, we can achieve a classification accuracy of 96% and higher per each class.

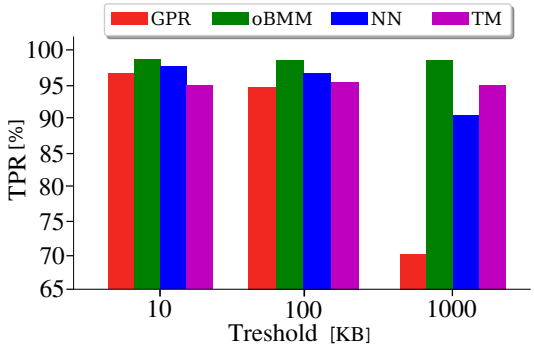


Fig. 6: True positive rate comparison

VI. Conclusion

In this work, we first show the usefulness of per-flow PSD in describing the flow behaviour. Then, we applied TM to classify flows by sizes, based on their packet size distribution that can be extracted from the first few packets of each flow. Based on the obtained results, packet size distribution can be used to describe the behaviour of the flow. The TM technique can achieve an overall accuracy of 96%.

This approach does not require any modification to the applications or end hosts and it provides an indication of which flow is leading to be a HH upon the start of each flow. The ability to predict which flow will have an impactful size in an early stage allows us to improve activities related to the network such as routing mechanisms, QoS provisioning, and so on. In particular, for the routing mechanisms, it helps to avoid congestion and to mitigate the need for load balancing.

However, the time required to collect enough flow details for reliable classification can still be a challenge and requires further research. Motivated by the results provided in this paper, future work will be aimed at investigating the usefulness of per-flow PSDs with other approaches that were shown to be effective in network traffic management [27], [28].

Acknowledgement

A. Duque-Torres is supported by the ISIF Internet Operations Research Grant (project #E3164). A. Pekar and W. Seah are supported by VUW's Huawei NZ Research Programme, Software-Defined Green Internet of Things (project #E2881).

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] A. Pekár, M. Chovanec, L. Vokorokos, E. Chovancová, P. Feciľak, and M. Michalko, "Adaptive aggregation of flow records," *Computing and Informatics*, vol. 37, no. 1, pp. 142–164, 2018.

[3] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications," *J. of Network and Computer Applications*, vol. 36, no. 2, pp. 567–581, 2013.

[4] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Internet Measurement Conference (IMC)*, Melbourne, Australia: ACM, 2010, pp. 267–280.

[5] L. Yang, B. Ng, and W. K. G. Seah, "Heavy hitter detection and identification in software defined networking," in *Proc. 25th International Conference on Computer Communication and Networks (ICCCN)*, Waikoloa, HI, USA, Aug. 2016, pp. 1–10.

[6] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "Devoflow: Cost-effective flow management for high performance enterprise networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets)*, Monterey, California: ACM, 2010, 1:1–1:6.

[7] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *Proc. 30th IEEE Int. Conf. on Computer Communications (INFOCOM)*, Apr. 2011, pp. 1629–1637.

[8] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings Symposium on SDN Research*, Association for Computing Machinery, Inc, Apr. 2017, pp. 164–176.

[9] M. Chiesa, G. Kindler, and M. Schapira, "Traffic engineering with equal-cost-multipath: An algorithmic perspective," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 779–792, Apr. 2017.

[10] R. Trestian, G. Muntean, and K. Katrinis, "Micetrap: Scalable traffic engineering of datacenter mice flows using openflow," in *Proc. 21st Int. Symp. on Integrated Network Management (IFIP)*, May 2013, pp. 904–907.

[11] R. Liu, H. Gu, X. Yu, and X. Nian, "Distributed flow scheduling in energy-aware data center networks," *IEEE Commun. Lett.*, vol. 17, no. 4, pp. 801–804, Apr. 2013.

[12] C. Lin, C. Chen, J. Chang, and Y. H. Chu, "Elephant flow detection in datacenters using openflow-based hierarchical statistics pulling," in *Proc. 33rd IEEE Global Communications Conference (GLOBECOM)*, Dec. 2014, pp. 2264–2269.

[13] C. Bi, X. Luo, T. Ye, and Y. Jin, "On precision and scalability of elephant flow detection in data center with sdn," in *Proc. 32nd IEEE Global Communications Conference (GLOBECOM) Workshops*, Dec. 2013, pp. 1227–1232.

[14] P. Poupart, Z. Chen, P. Jaini, F. Fung, H. Susanto, Y. Geng, L. Chen, K. Chen, and H. Jin, "Online flow size prediction for improved network routing," in *Proc. IEEE ICNP Workshop on Machine Learning in Computer Networks (NetworkML)*, Singapore, Nov. 2016, pp. 1–6.

[15] Z. Liu, D. Gao, Y. Liu, H. Zhang, and C. H. Foh, "An adaptive approach for elephant flow detection with the rapidly changing traffic in data center network," *Int. J. of Network Management*, vol. 27, no. 6, e1987, Jul. 2017, e1987 nem.1987.

[16] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2037–2064, 2014.

[17] A. Duque-Torres, A. Pekar, W. K. G. Seah, and O. M. C. Rendon, "Clustering-based analysis for heavy-hitter flow detection," in *Asia Pacific Regional Internet Conference on Operational Technologies (APRICOT)*, Daejeon, South Korea, Feb. 2019.

[18] A. Gorcin and H. Arslan, "Template matching for signal identification in cognitive radio systems," in *Proc. IEEE Military Communications Conference (MILCOM)*, Oct. 2012, pp. 1–6.

[19] B. W. Silverman, *Density estimation for statistics and data analysis*. Routledge, 2018.

[20] M. Rousson and D. Cremers, "Efficient kernel density estimation of shape and intensity priors for level set segmentation," in *Proc. International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2005, pp. 757–764.

[21] S. Zhong, T. M. Khoshgoftaar, and N. Seliya, "Analyzing software measurement data with clustering techniques," *IEEE Intelligent Systems*, vol. 19, no. 2, pp. 20–27, 2004.

[22] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*, Springer, 2009, pp. 1–4.

[23] J. Zhao and L. Itti, "Shapedtw: Shape dynamic time warping," *Pattern Recognition*, vol. 74, pp. 171–184, 2018.

[24] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python*, Available: http://www.scipy.org/, 2001.

[25] W. Meert and T. V. Craenendonck, *Wannesm/dtaidistance v1.1.2*, Available: https://github.com/wannesm/dtaidistance, Jul. 2018.

[26] K. J. Cios, W. Pedrycz, R. W. Swiniarski, and L. A. Kurgan, *Data Mining: A Knowledge Discovery Approach*. Berlin, Heidelberg: Springer-Verlag, 2007.

[27] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. Caicedo Rendon, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, May 2018.

[28] S. Ayoubi, N. Limam, M. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. Caicedo Rendon, "Machine learning for cognitive network management," *IEEE Communications Magazine*, vol. 56, Jan. 2018.