IET Information Security

Research Article

IET Journals
The Institution of Engineering and Technology

Inline high-bandwidth network analysis using a robust stream clustering algorithm

ISSN 1751-8709
Received on 28th November 2017
Revised 12th February 2019
Accepted on 4th March 2019
E-First on 24th April 2019
doi: 10.1049/iet-ifs.2018.5287
www.ietdl.org

Morteza Noferesti[1], *Rasool Jalili*[1] ✉

[1]*Data and Network Security Lab, Sharif University of Technology, Tehran, Iran*

✉ *E-mail: jalili@sharif.edu*

Abstract: High-bandwidth network analysis is challenging, resource consuming, and inaccurate due to the high volume, velocity, and variety characteristics of the network traffic. The infinite stream of incoming traffic forms a dynamic environment with unexpected changes, which requires analysing approaches to satisfy the high-bandwidth network processing challenges such as incremental learning, inline processing, and outlier handling. This study proposes an inline high-bandwidth network stream clustering algorithm designed to incrementally mine large amounts of continuously transmitting network traffic when some outliers can be dropped before determining the network traffic behaviour. Maintaining extended-meta-events as abstracting data structures over a sliding window, enriches the algorithm to address the high-bandwidth network processing challenges. Evaluating the algorithm indicates its robustness, efficiency, and accuracy in analysing high-bandwidth networks.

1 Introduction

Numerous network services, from real-time user monitoring to providing network security, require real-time high-bandwidth network traffic analysis. Growing in size and complexity of networks makes administrators using more practical and intelligent network traffic analysis tools to maintain the network more stable, available, secure, and well-planned. Continuous increasing of applications in high-bandwidth networks, which leads to high-volume, velocity, and variety of traffics, initiates new technical challenges and requirements in traffic analysis systems.

The port-based analysis is the first classical method, which analyses the network traffic based on the well-known port numbers. Moore and Papagiannaki [1] claim that port-based methods have no better accuracy than 70%. Madhukar and Williamson [2] show that port-based methods are unable to identify 30–70% of internet traffic flows they investigated. In a recent study [3], Alcock *et al*. quantify network traffic on major transmission control protocol (TCP) and user datagram protocol (UDP) ports and confirm that port-based identification is ineffective.

An alternative to port-based analysis approaches is traffic packets inspection methods, known as deep packet inspection (DPI) techniques. Such methods infer application types by looking for application-specific data within traffics' payload. These approaches rely on two related assumptions, visibility of payloads and being known the syntax of application packets payload. Although DPI-based analysis is claimed to achieve high accuracy, the assumptions are not valid in real situations. The visibility of payloads assumption can be violated when the traffic is encrypted. The second assumption produces a heavy operational load, especially in high-bandwidth networks. Moreover, updating and changing in each application reduce accuracy. Given the high resource requirements, currently, DPI-based methods are used to analyse network traffic in small networks and to establish accurate ground-truth data.

The low accuracy of port-based analysis methods and the high overhead of DPI-based analysis methods have motivated the use of machine-learning techniques for traffic analysis [4]. Current machine-learning-based methods can be structured into supervised and unsupervised. The supervised methods detect characteristic patterns of different applications, utilising a labelled dataset [5]. The unsupervised learning approaches cluster traffic flows into groups that have similar traffic patterns.

Clustering traffic is widely utilised in application-specific services, such as efficient network resource management [6] or applying different tariffs for provided services [5], as well as the network security domain, such as automated intrusion detection systems [7] or detection of denial of service attacks [8]. Basically, three main clustering methods have been used in the network traffic clustering literature [5, 9–11]: (i) the classic k-means algorithm that partitions instances into disjoint clusters, (ii) the clustering algorithms that generate a hierarchical grouping of instances, (iii) the probability based method assigns instances into classes probabilistically, not deterministically.

High-bandwidth networks are dynamic environments having real-time incoming data forming a potentially infinite stream with unexpected changes. Such environments require adaptive analysing approaches which learn the high-bandwidth network traffic dynamic behaviours. The analysing approaches should satisfy the following [12–14]:

- *Incremental learning*: The methods should be periodically updated to adapt to new applications.
- *Outlier handling*: High-bandwidth networks tend to be noisy due to the lack of prior processing. Analysing methods should have strategies to detect and resolve outliers.
- *Forgetting mechanism*: Analysing methods have to forget the expired learning information.
- *Inline processing*: Unbounded high-bandwidth network traffic imposes that each packet to be processed once (sometimes called one-pass processing restriction).

In this study, the high-bandwidth network traffic is modelled as a massive and unbounded sequence of flows, continuously generated at rapid rates, called a flow stream (FS) [15–17]. Successive packets having the same five-tuple 〈source-IP, source-port, destination-IP, destination-port, TCP/UDP protocol〉 construct a flow. A FS is defined as

Definition 1: A FS is a potentially infinite ordered sequence of flows, denoted by $\langle \boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_n \rangle$ where all flows $\boldsymbol{f}_i \in \text{FS}$ follow the schema $\boldsymbol{f}_i = \langle a_1, \ldots, a_d \rangle$ and $a_1, \ldots, a_d$ are flow attributes in a d-dimensional space.

The FS clustering problem is defined as maintaining a consistent set of clusters contain the sequence of observed flows so far, in a bounded amount of memory and time. More formally:

Definition 2: Suppose that $FS_t = \langle \boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_t \rangle$ is a sequence of observed flows of an input FS at time-stamp t. A stream clustering algorithm should find a set of clusters $\text{cluster}_t = \{C_1^t, C_2^t, \ldots, C_n^t\}$, where $\cap_{i=1}^{n} C_i^t = \varnothing$ and $\cup_{i=1}^{n} C_i^t = FS_t$.

Considering the high-bandwidth network traffic as a dynamic environment having a potentially infinite stream of flows, this study proposes a high-bandwidth network stream clustering algorithm (HBN-SCA) as a window-based stream clustering algorithm to tackle high-bandwidth networks analysing challenges such as incremental learning, outlier handling, forgetting mechanism, and inline processing. Under the sliding window model, the algorithm maintains statistics and information about the recent traffic in an abstract data structure called extended-meta-event (EM). The EMs are used to increase the clustering accuracy and robustness against traffic outliers. Experimental results have empirically demonstrated that developing HBN-SCA yields high accuracy and efficiency in real network traffic and workloads.

1.1 Contributions

The main contribution of the study is to cluster the high-bandwidth network traffics in a robust setting where outliers are detected in a systematic approach. To handle outliers properly, HBN-SCA models the traffic of the high-bandwidth network as a noisy stream in a formal way and uses the algorithm proposed by Zarrabi-Zadeh *et al*. [18] to analyse the input stream where the specific number of outliers can be dropped from the stream before analysing.

Other contributions of this study are as follows:

- The main difficulty in handling outliers in the data stream model is the absence of any information about the incoming flows in the future. We define the EM structure as a summarisation of a set of d-dimensional flows based on the meta-event data structure, defined in [14]. The EMs have a notion of a buffer, intended to handle outliers in the algorithm. The outliers would be caught in the buffer or left outside of the already existing EMs and put in a global buffer called window.
- Creating new EMs by finding its centre and radius is very important in the accuracy and performance of the algorithm. HBN-SCA uses the density-based spatial clustering of applications with noise (DBScan) algorithm [19] to create new EMs based on the density of flows in the window. Accordingly, HBN-SCA detects core-flows in the window and dynamically finding the EM radius.

1.2 Organisation

The rest of this paper is organised as follows: an overview of practical methodologies employed in high-bandwidth network analysing is given in Section 2. HBN-SCA is defined in Section 3. Empirical simulation and real-network implementation of HBN-SCA are explained in Section 4 where it is demonstrated that HBN-SCA produces accurate clusters regarding the input FS. Finally, the on-going challenges and future directions are discussed in Section 5.

2 Related works

Related works are surveyed in two subsections. Firstly, the state-of-the-art in high-bandwidth network analysing is discussed. Then, the current stream clustering algorithms are reviewed.

2.1 High-bandwidth network analysing approaches

The specific application of machine-learning techniques to traffic classification has attracted large attention from the research community [9, 11]. Perera *et al*. [5] surveyed the literature of machine-learning techniques in the field of network traffic classification. They compared the performance of six widely-used supervised machine-learning algorithms such as multilayer perceptron, decision tree, random forest, Bayes networks, naive Bayes, and Bayes tree for classifying network traffic. The evaluations were conducted for the classification of five distinct network traffic classes and two feature selection techniques. Their results indicated that the random forest and decision tree algorithms are promising classifiers for network traffic in terms of both classification accuracy and computational efficiency.

In recent years, continuous increasing of network bandwidth dictates new requirements to be addressed in network analysing methods. The U.S. patent [20] focusing on real-time network monitoring and security concludes that security systems such as firewalls, anti-viruses, and intrusion detection systems require real-time high-bandwidth network processing. While the patent provides the commercial impacts of research about real-time high-bandwidth network monitoring and security; it advises that the full-line rate of a high-bandwidth network corresponds to rates above 100 Mbps and in some cases above 1000 Mbps. Noferesti and Jalili [14] proposed a behaviour detection system, namely high-bandwidth network behaviour detection system (HB^2DS), where a data structure called meta-event is defined whose inter-relationships are used to mine end-user behaviours. Authors represented a formal definition of a behaviour detection system and demonstrated that HB^2DS satisfies the behaviour detection system constraints.

One group of researchers attempts to distinguish unknown applications in high-bandwidth networks. Zhang *et al*. [4] proposed the robust traffic classification method by combining supervised and unsupervised learning techniques to identify unknown applications. In [21], Wang *et al*. proposed the unclean traffic classification (UTC) method, which filters unclean training samples. UTC applies multiple classifiers in the training phase. If most classifiers mark a data as noise, it will be removed from the training data. Lin *et al*. [22] proposed an unknown network protocol classification method based on semi-supervised learning. They apply the K-means algorithm over training data. For each cluster, if it contains no labelled sample, it is detected as unknown.

The other group of researchers attempts to increase the performance of machine-learning methods in network analysis. Arora *et al*. [23] used Netflow to categorise network nodes in a data centre. Based on Netflows' information, nodes are divided into either the client or the server. Lu *et al*. [24] used message size sequence and distribution to classify some popular network protocols, such as file transfer protocol (FTP) or hypertext transfer protocol (HTTP), in high-bandwidth networks.

Gomes *et al*. [25] surveyed the network traffic analysis approaches using ensemble learning methods. Utilising these methods, Casas *et al*. [12] combined multiple DBScan algorithms to increase the network traffic clustering accuracy. They divided the feature space into sub-spaces and applied the DBScan algorithm on each subspace. Then, a voting algorithm distinguished anomalies in network traffic. Jin *et al*. [13] presented a series of simple linear binary classifiers, each of which can be efficiently implemented and trained in parallel.

2.2 Stream clustering algorithms

Data stream mining is an active research area that has recently emerged to discover knowledge from large amounts of continuously generated data [26, 27]. Major data stream clustering approaches can be categorised into partitioning algorithms, micro-clustering algorithm, grid-based algorithms, model-based algorithms, and density-based algorithms. Density-based clustering can discover the clusters of non-spherical shape, filter outliers, and have no assumption on the number of clusters [28]. So, density-based clustering is more appropriate for high-bandwidth network clustering and is reviewed in this section.

D-Stream [29] is an algorithm for clustering data streams based on the density-based approach. It has online and offline phases. The online phase maps the data into some grids and the offline phase clusters the grids based on their densities. It uses fading window model to capture the evolution of clusters. It assigns a weight to each data point and determines grid density based on these weights. D-Stream forms a cluster based on high-density grids which are called dense grids. Final clusters are formed by

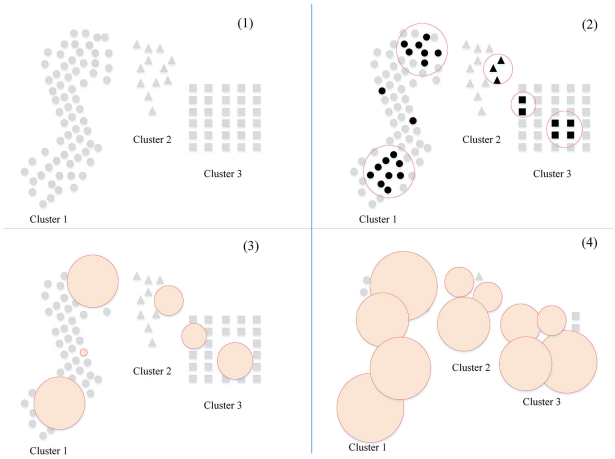

Fig. 1 *Example of the online abstraction step of HBN-SCA*

merging dense grids. Furthermore, it has a technique for removing the sporadic grids mapped by outliers.

DenStream [30] is also a density-based method for clustering an evolving data stream. It has online and offline phases; the online phase uses core-micro-clusters, potential-micro-clusters, and outlier micro-clusters to summarise clusters with arbitrary shapes and distinguish potential outliers. The offline phase clusters the potential micro-cluster based on DBScan. Moreover, a pruning strategy is designed to remove the outlier micro-cluster periodically.

CluStream [26] is designed to cluster data streams over different time horizons in an evolving environment. Since the snapshots of clustering results over the landmark window have to be stored, CluStream consumes more space, and so is not suitable for the clustering problem over sliding windows. In addition, the influence of expired records cannot be promptly eliminated during on-line clustering, while such a prompt elimination is required in sliding window clustering.

Aggarwal [15] extended cluster feature for sliding window model to present cluster distribution changes as well. The proposed approach keeps aggregates over the sliding window and calculates clustering based on all exponential histogram cluster feature synopsis. The approach cannot discover clusters with arbitrary shapes and does not have the ability to handle outliers. SDStream [31] extends it for density-based clustering. The algorithm uses the potential and outlier micro-cluster concepts and keeps them in the form of an exponential histogram. SDStream removes the outlier micro-clusters, which are not in the sliding window.

While state-of-the-art in network traffic analysis try to improve accuracy and/or performance, the dynamic nature of high-bandwidth networks is not considered. Clustering of dynamic network traffic requires that a process be able to incrementally cluster flows within memory and time restrictions [16]. In addition to evaluating the performance of stream clustering algorithm in analysing network traffic, this study proposes HBN-SCA as a high-bandwidth network specific clustering algorithm which has incremental learning, outlier handling, forgetting mechanism, and inline processing properties.

3 High-bandwidth network stream clustering algorithm (HBN-SCA)

To satisfy the HBN analysing requirements, we introduce HBN-SCA. Considering $\mathrm{FS} = \{\boldsymbol{f}_1, \boldsymbol{f}_2, \ldots, \boldsymbol{f}_n\}$ as a high-bandwidth network FS, O as the set of outliers, and $P = \mathrm{FS} \setminus O$ as the set of non-outliers. In [14], the authors use the well-known minimum enclosing ball problem to maintain network traffic behaviours. The problem is to fit a ball over FS, where each flow processed once. The approximation algorithm finds the region $R(\mathrm{FS})$ enclosing flows in FS which is $\frac{3}{2}$ factor of the optimum area $R^*(\mathrm{FS})$. Considering outliers in the FS, an $\alpha\beta$-approximation algorithm for the problem satisfies the followings:

$$R^*(\mathrm{FS}) \leq \alpha R(\mathrm{FS}) \leq \alpha\beta R(P)\,. \tag{1}$$

The upper bound for α is $\frac{3}{2}$ [14]. HBN-SCA uses the algorithm proposed in [18] to provide some upper bounds on β for high-bandwidth networks with outliers. The main difficulty in handling outliers in the data stream model is the absence of any information about the incoming flows in the future. HBN-SCA uses a buffering framework, which postpones the decision about the input flows until it gets enough information. The buffering framework is based on the EM data structure which is defined as follows:

Definition 3: An EM is a summarisation of a set of d-dimensional flows $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n\}$; which is defined as a tuple $(\boldsymbol{c}, r, b, n, t)$, where $\boldsymbol{c}$ (the meta-event centre) corresponds to a vector of d-entries, r (radius) is the maximum distance of flows from $\boldsymbol{c}$, b is a buffer containing at most m flows of the stream, n is the number of flows, and t is a time-stamp declaring the last update of the EM. For each dimension, the average of attributes is maintained in $\boldsymbol{c}$. The kth entry of $\boldsymbol{c}$ is equal to $(\sum_{i=1}^{n}(f_i^k))/n$, where (f_i^k) is the value of $\boldsymbol{f}_i$ in the kth dimension.

To cope with the non-deterministic nature of the input stream, the EM data structure has two important properties:

1. *Incrementality*: An incoming flow $\boldsymbol{f}$ can be easily inserted into an EM by updating its attributes. Considering $\mathrm{dst}(\cdot)$ as a distance function and $\sigma = 1/2(\mathrm{dst}(\boldsymbol{f}, \mathrm{EM}\cdot\boldsymbol{c}) - \mathrm{EM}\cdot r)$, the new radius and centre of EM can be calculated according to (2) and (3)

$$\mathrm{EM}\cdot r = \mathrm{EM}\cdot r + \sigma, \tag{2}$$

$$\mathrm{EM}\cdot\boldsymbol{c} = \boldsymbol{f} + \frac{\mathrm{EM}\cdot r}{\mathrm{EM}\cdot r + \sigma} \times \mathrm{dst}(\boldsymbol{f}, EM\cdot\boldsymbol{c})\,. \tag{3}$$

2. *Additivity*: Two disjoint EMs, EM_i and EM_j, can be easily merged into EM_k. In [14], it is shown that EM_k is obtained by inserting a flow $\boldsymbol{f}_t$ into EM_i regarding the incrementality property, where

$$\boldsymbol{f}_t = \mathrm{EM}_i + \frac{\mathrm{EM}_i\cdot r}{\mathrm{dst}(\mathrm{EM}_i\cdot\boldsymbol{c}, \mathrm{EM}_j\cdot\boldsymbol{c})} \times (\mathrm{EM}_i\cdot\boldsymbol{c} - \mathrm{EM}_j\cdot\boldsymbol{c})\,.$$

HBN-SCA consists of two main steps: online abstraction and offline conclusion. The online abstraction step maintains a set of EMs $\mathrm{EMS}_t = \{\mathrm{EM}_1^t, \ldots, \mathrm{EM}_m^t\}$ at time-stamp t. It has two main modules, namely *Learning* and *Window*. The Learning module processes the input FS inline and incrementally updates the stream status in EMS_t. The Window module defines new EMs over the current state of the input stream. The Conclusion module in the offline abstraction step constructs clusters set $\mathrm{cluster}_t = \{C_1^t, \ldots, C_n^t\}$ using EMS_t.

3.1 Online abstraction

HBN-SCA summarises incoming flows in an online abstraction step. Fig. 1 shows an example of the abstraction step procedure. The total stream is depicted in part (1). In part (2), flows which are indicated by black colour, are inserted into the algorithm. As the figure shows, HBN-SCA tries to cover the new arriving flows with EMs (circles). As the stream proceeds, part (3), some EMs are merged (the additivity property), and some are updated regarding the arriving flows (the incrementality property). Regarding these two properties, HBN-SCA can determine the arbitrary-shaped clusters. In the next two subsections, first, we show how the Learning module processes the input FS and incrementally update the abstraction. Then, the details of the Window module will be discussed.

for each Flow f in FS **do**
  if there exists an $EM \in EMS_t$ which covers f **then**
    insert f into $EM.b$;
    if $EM.b$ is full **then**
      find the closest flow $f_{selected} \in EM.b$ to the center-point of $EM.b$;
      insert $f_{selected}$ into EM according to the incrementality property of extended-meta-events;
      update $EM.t$;
    end if
  else
    insert f into the window W and mark it as $unknown$;
    if W is full **then**
      create a set of extended-meta-events as $TempExtendedMetaEvent$;
      call $ProcessWindow(W, TempExtendedMetaEvent, Outlier_{THR}, eps$,*min-Point*, $StepSize)$;
      insert $TempExtendedMetaEvent$ into EMS_t according to the additivity property of extended-meta-events;
      free W;
    end if
  end if
  call the forgetting mechanism every $Expired_{THR}$ duration;
end for

Fig. 2 *Algorithm 1: HBN-SCA (FS,* EMS_t*, W,* $\text{Outlier}_{\text{THR}}$*,* $\text{Expired}_{\text{THR}}$*, min-Point, eps, StepSize)*

adapt eps with $StepSize$ and find core-flows in W according to eps and *min-Point* using DBScan algorithm;
for each core-flow, define an extended-meta-event $EM_{new} = (\vec{c}_{new}, r_{new}, b_{new}, n_{new}, t_{new})$ where $\vec{c}_{new}$ is the core-flow, $r_{new} = eps$, $b_{new} = sizeof(W)/sizeof(\text{EMS}_t)$, $n = 0$, and t is the current time-stamp;
insert EM_{new} into $TempExtendedMetaEvent$ set;
for each flow $\vec{f}_{window}$ in W **do**
  if there exists an $EM \in TempExtendedMetaEvent$ which covers $\vec{f}_{window}$ **then**
    remove $\vec{f}_{window}$ from W;
    insert flow $\vec{f}_{window}$ into EM;
  else
    if the waiting time of $\vec{f}_{window} \geq Outlier_{THR}$ **then**
      remove $\vec{f}_{window}$ from the window and mark it as outlier;
    else
      increase $\vec{f}_{window}$'s waiting time;
    end if
  end if
end for

Fig. 3 *Algorithm 2: ProcessWindow (W,* $\text{Outlier}_{\text{THR}}$*, eps, min-Point, TempExtendedMetaEvent, StepSize)*

3.1.1 Learning module: HBN-SCA maintains an EM set, $\text{EMS}_t = \{\text{EM}_1^t, \ldots, \text{EM}_m^t\}$, which contains all the EMs detected by the algorithm until time-stamp t. For each input flow, HBN-SCA decides to insert it into an EM buffer or to put it into the window. If the input flow inserted into the buffer and it is made full, the algorithm extracts one flow from the buffer and updates the EM. Now, the problem is how to extract flows from the buffer in such a way that only outliers remain in the buffer. In an ideal scenario, all outliers are kept in the buffer, and only non-outlier flows are extracted. However, this is impractical in real networks as outliers and non-outliers are not precisely distinguishable.

HBN-SCA provides a mechanism to bind the total error based on the algorithm proposed in [18]. The algorithm uses the concept of centre-point, which is defined as follows:

Definition 4: Considering an n-point set P in d dimensions, a point $c \in R^d$ is called a centre-point of P if any half space containing c contains at least $\lceil n/(d+1) \rceil$ points of P.

In other words, any half space that avoids a centre-point can contain at most $n - n/(d+1) = dn/(d+1)$ flows of FS. Considering FS with size $(d+1)(z+1)$ where z is the number of outliers, and c is the centre-point of FS, regarding Definition 4, any region that avoids c contains at most the following number of flows:

$$\lfloor d|P|/(d+1) \rfloor = d(z+1)\,. \tag{4}$$

From the other side, the number of non-outliers in FS is at least:

$$|\text{FS} \setminus O| = |P| = (d+1)(z+1) - z = d(z+1) + 1. \tag{5}$$

Regarding (4) and (5), c is contained in any region than encloses non-outliers, and in particular, is contained in the final minimum enclosing region. So, let b be the buffer of an EM, if $|b| \geq (d+1)(z+1)$, HBN-SCA extracts from b a flow which is closest to the centre-point of b. Zarrabi-Zadeh and Mukhopadhyay [18] proved that this algorithm yields an approximation factor of $\sqrt{2}$. To achieve the approximation factor, the buffer size (m in Definition 3) should be more than $(d+1)(z+1)$.

The details of the HBN-SCA learning module is described in Algorithm 1 (see, Fig. 2). HBN-SCA reads flows continuously from the FS. All the EMs detected by the algorithm is stored in a set called EMS_t. W is the window data structure. The value of $\text{Outlie}_{\text{THR}}$ and $\text{Expired}_{\text{THR}}$ represent the outlier detection and expired threshold. min-Point, eps, and StepSize are used in the window processing algorithm.

According to Algorithm 1 (Fig. 2), if there exists an EM in EMS_t which covers the flow (the flow distance to the EM centre is less than the EM radius), it is inserted into the EM buffer. When the buffer becomes full, the algorithm finds the buffer's centre point according to Definition 4. Then, the closest flows to the centre-point in the buffer are inserted into the EM. If there is not any EMs covering the input flow, the flow is inserted into the window (W). When the window becomes full, ProcessWindow algorithm is invoked [Algorithm 2 (see Fig. 3)] to generate new EMs, which are inserted into EMS_t. Finally, the window W set free.

The HBN-SCA forgetting mechanism specifies how old data are discarded. It is a trade-off between the reactivity of the system and robustness to outliers. HBN-SCA uses a gradual forgetting mechanism based on a linear decay technique. EMs have a time-stamp shows its last update (e.g. insert a new flow or merge with another EMs). If one EM does not update for more than the $\text{Expired}_{\text{THR}}$ threshold, it is expired and removed from EMS_t.

3.1.2 Window module: To summarise the continuously-arriving flows and, at the same time, giving the greater importance to up-to-date ones, a time window is defined in HBN-SCA, which covers the most recent flows, which are not covered by EMs. More formally, the window W consists of a set of flows $\{\boldsymbol{f}_1, \ldots, \boldsymbol{f}_w\}$, where all flows $\boldsymbol{f}_i \in W$ are not assigned to an EM in EMS_t. The window model is similar to the sliding window model, which is widely adopted in stream mining [25]. HBN-SCA creates new EMs based on the existing flows in W. It also detects outliers according to the time quantum that flow remains in the window. To process the window, the concept of core-flow is defined as follows:

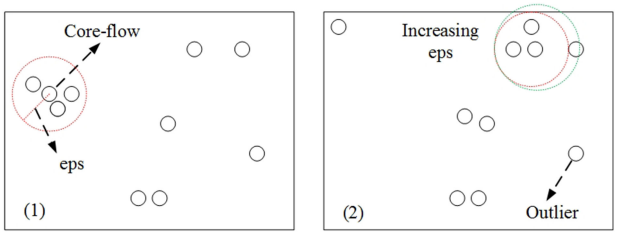

Fig. 4 *Different flows distributions in a window*

*Definition 5: (core-flow)***:** *For an eps and a min-Point value, a flow* $\boldsymbol{f}$ *in a window W is a core-flow, if there exists at least a minimum number (min-Point) of flows such as* $\boldsymbol{f}_{\text{sam}}$ *in W where distance*$(\boldsymbol{f}_{\text{sam}}, \boldsymbol{f}) \leq eps$.

Fig. 4 presents the different flow distributions in a window when the stream proceeds. In the figure, the window size and min-Point values are 10 and 3, respectively. In part (1) of the figure, the algorithm finds a core-flow while there exists min-Point number of flows where their distance to the core-flow is less than eps value. Another distribution is shown in part (2), where the algorithm is unable to find a core-flow, so it increases the eps value until finding a core-flow. The increased ratio is determined by StepSize initial parameter.

The details of the window processing algorithm, ProcessWindow, is represented in Algorithm 2 (Fig. 3). The algorithm adapts eps and uses the DBScan algorithm to find core-flows. Then, it defines EMs for core-flows. To define an EM (EM_{new}), its parameters ($\boldsymbol{c}_{\text{new}}, r_{\text{new}}, b_{\text{new}}, n_{\text{new}}, t_{\text{new}}$) are initialised, where $\boldsymbol{c}_{\text{new}}$ is the core-flow, the initial r_{new} value is set to eps, b_{new} is set to the size of W divided by the size of EMS_t, n_{new} and t_{new} are set to zero and the current time-stamp, respectively. These new EMs are inserted into a set, namely TempExtendedMetaEvent.

For each flow in the window, if there exists an EM in TempExtendedMetaEvent, which covers it, the flow is removed from the window. If a flow remains in the window (there exist no similar flow to it) more than $\text{Outlier}_{\text{THR}}$ threshold, it is marked as an outlier. HBN-SCA detects outliers based on the time period they remain in the window. The threshold $\text{Outlier}_{\text{THR}}$ specifies the time threshold one flow can remain in the window. When the threshold is overcome, the flow is considered as an outlier. In Fig. 4 part (2), supposing $\text{Outlier}_{\text{THR}} = 1$, the flow is labelled outlier because it remains in a sparse region for two windows.

3.2 Offline conclusion

Utilising EMs as the summarised data structures generated in the online abstraction step, HBN-SCA tries to detect the related EMs and insert them into the same clusters. Suppose EM_i and EM_j as two EMs, if their distance is less than the sum of their radius, HBN-SCA inserts them into the same cluster. The offline conclusion step is periodically invoked, which produces convex or non-convex clusters over the current state of the input stream.

3.3 Complexity analyses

The complexity of the online and offline phases, as well as the complexity of window processing of HBN-SCA, are shown in the following theorems, where m is the number of EMs in $\text{EMS}_t = \{\text{EM}_1^t, \ldots, \text{EM}_m^t\}$, n is the number of clusters in the cluster set $\text{cluster}_t = \{C_1^t, \ldots, C_n^t\}$, and w is the window size.

Theorem 1: The time complexity of the process window algorithm is $O(\text{diameter}(W) \times w \times \log w)$.

Proof: The algorithm uses the DBScan algorithm over the window with size w, which has the time complexity $O(w \log w)$. HBN-SCA executes DBScan to adapt eps value. In the worst case, DBScan executes diameter (W)/StepSize times. diameter(W) is the diameter of the window. While StepSize is a constant value, the process window algorithm time complexity is $O(\text{diameter}(W) \times w \times \log w)$. □

Theorem 2: : The time complexity of the online abstraction phase is $O(m + diameter(W) \times w \times \log w)$.

Proof: The online abstraction phase contains two modules. The first finds the closest EM for each input flow, so its time complexity is $O(m)$, where m is the number of EMs. It also updates the EM where it requires computing of centre-points of $O(dz)$ flows in d dimensions which is polynomial in d and z. The second module processes window algorithm where its complexity is $O(\text{diameter}(W) \times w \times \log w)$. As a result, the time complexity of online abstraction phase is $O(m + \text{diameter}(W) \times w \times \log w)$. □

Theorem 3: The time complexity of the offline conclusion phase is $O(m^2)$.

Proof: The offline conclusion tries to detect the related EMs and inserts them into the same cluster of cluster_t. So, the algorithm should compare all the entities in the EM set pairwise. That sets the time complexity of the offline conclusion phase to $O(m^2)$. □

4 Evaluation

To evaluate our proposed framework in terms of analysing high-bandwidth networks, we developed HBN-SCA and compared its results with HB^{2}DS [14], DBStream [32], and DenStream [33]. The approaches are implemented in C++ (gcc version 5.2.1) on Intel Xeon E5620 CPU having Linux kernel version 3.17.0.3, and 42 GB of main memory. To read packets directly from the interface drivers, we used the PF_RING library [34] (version 6.4.1), which provides direct access to traffic packets without the kernel intermediation.

4.1 Evaluation criteria

To evaluate the quality of detected clusters, four evaluation criteria are used: purity, Rand index (RI), sum of squared errors (SSE), and Silhouette. Purity and RI parameters are used to measure the similarity of flows inside a cluster. SSE parameter and Silhouette coefficient represent the similarity of a flow to its own cluster compared to the other clusters.

Considering $\text{Dom}(C_i)$ as the dominant class label of flows belonging to cluster C_i, purity calculates the ratio of flows which are assigned to clusters where the clusters Dom(.) values are equal to the flows class labels. Purity is computed through (6), where DominantFlowRatio(.) counts the number of flows in an EM with dominant class label divided by the number of EMs in the cluster

$$\text{Purity} = \frac{1}{m} \sum_{i=1}^{m} \sum_{\text{EME}_k \in C_i} \text{Dominant Flow Ratio}\,(\text{EME}_k)\,. \tag{6}$$

RI measures the percentage of assignments that are correct. An assignment is correct if it assigns similar flows to similar clusters (true positive), or it assigns dissimilar flows to different clusters (true negative). It can be formulated by (7), where N is the number of flows in the input stream, and $\sum \text{TP}$ and $\sum \text{TN}$ are the number of true positive and true negative assignments, respectively

$$\text{RI} = \frac{\sum \text{TP} + \sum \text{TN}}{N}\,. \tag{7}$$

The compactness of clusters is evaluated through SSE criterion, where the lower SSE value shows the more compactness of clusters. It can be formulated by (8), where m is the number of clusters

$$\text{SSE} = \sum_{i=1}^{m} \sum_{\text{EME}_k \in C_i} \sum_{f_j \in \text{EME}_k} \text{distance}(\text{EME}_i, f_j)^2\,. \tag{8}$$

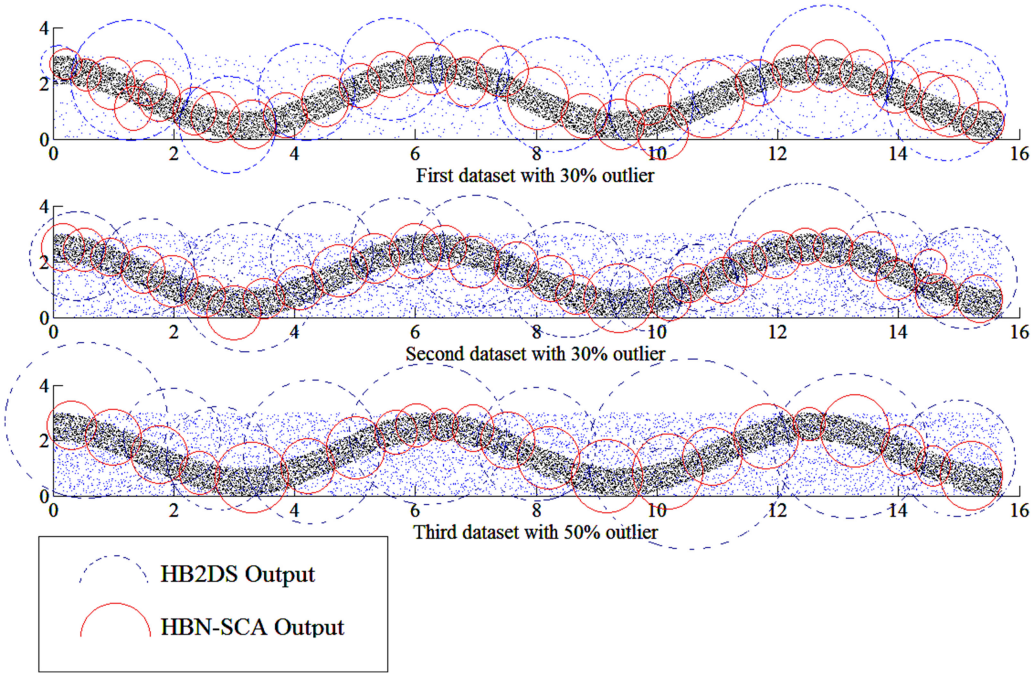

Fig. 5 *HBN-SCA and HB^{2}DS outputs scatter plots on the synthetic datasets*

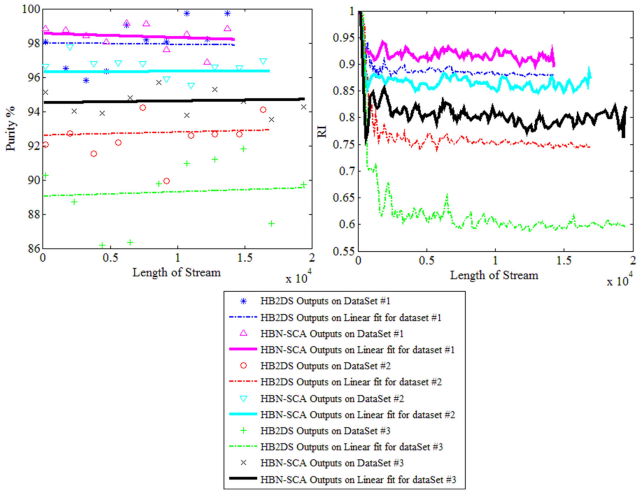

Fig. 6 *HBN-SCA and HB^{2}DS evaluation parameters on the synthetic datasets*

The Silhouette value is a measure of how similar a flow is to its own cluster compared to the other clusters. Let dst($\boldsymbol{f}_i, C_b$) be the average distance between the flow $\boldsymbol{f}_i$ and all flows within the same cluster C_b, and dst($\boldsymbol{f}_i, C_e$) be the lowest average distance of $\boldsymbol{f}_i$ to all flows in any other cluster except C_b, the Silhouette value of flow f_i is computed by (9). Silhouette value of the stream is the average Silhouette values of all flows

$$\text{Silhouette}(\boldsymbol{f}_i) = \frac{\text{dst}(\boldsymbol{f}_i, C_e) - \text{dst}(\boldsymbol{f}_i, C_b)}{\max(\text{dst}(\boldsymbol{f}_i, C_b), \text{dst}(\boldsymbol{f}_i, C_e))}. \quad (9)$$

4.2 Synthetic dataset

To compare HBN-SCA with HB^{2}DS, HBN-SCA is applied on synthetic datasets first defined in [14]. The synthetic datasets contain 13,000 events, which are creating a sinusoidal-shaped behaviour with 10, 30, and 50% level of outliers, which have a uniform distribution. In the experiment, default values for HBN-SCA are $W = 2000$, OutlierThr = 6, eps = 0.2, min-Point = 10, and StepSize = 0.5.

HBN-SCA detected clusters are compared with HB^{2}DS results reported in [14] through Fig. 5. HBN-SCA detected the single sinusoidal-shaped cluster in each dataset. Comparing with the cluster found by HB^{2}DS (the blue-dotted circles), the cluster detected by HBN-SCA (the red circles) is smaller and more fit over the sinusoidal-shaped cluster. As a result, the error rate in HBN-SCA is decreased which is illustrated in Fig. 6.

As Fig. 6 shows, the purity of cluster detected by HB^{2}DS around 10% decreased while the outlier percentage increased. However, the decreased ratio in HBN-SCA is about 4%, because it is more robust against outliers. To fit a straight line over the purity values, the linear regression model is used while the points in the figure are some samples of the purity values. The RI value for HBN-SCA cluster is also more robust against the outliers percentage.

4.3 Comparing different stream clustering algorithms

In a pilot study, we examined HBN-SCA to evaluate its capability for clustering high-bandwidth real network traffic and compare it with the stream clustering algorithms HB^{2}DS, DBStream, and DenStream. The network traffic of an enterprise institute for four weeks which resulted in 2 TB stored raw traffic, which is fed into the clustering algorithms with 10 Gbps speed. We used the extensive R libraries, including stream [35] and streammoa [36], to implement and compare the aforementioned stream clustering algorithms with HBN-SCA.

The packets are forwarded to the nDPI library version 1.8 [37] (nDPI is an open source deep packet inspection tool that specifies the session protocol). The nDPI results are used as the ground truth in the clustering evaluation process. The network traffic includes a wide range of applications, while nDPI detected more than 59 different applications such as DNS, network basis input/output system (NetBIOS), SAMBA, HTTP, secure shell (SSH), real-time messaging protocol (RTMP), simple mail transfer protocol (SMTP), post office protocol version 3 (POP3), and network time protocol (NTP). About 3.5×10^5 flows are labelled by the nDPI module. After every 10,000 flows, the evaluation measures are calculated. Each clustering algorithm is evaluated against different initial parameters. Total results are represented in the Appendix.

28 features are extracted for each flow. The features are shown in Table 1. As the quality of the feature list is crucial to the accuracy of the clustering algorithm, in this study, we use the feature list proposed by Wang *et al.* [38]. We only use both directions traffic features, which are appropriate for TCP sessions. In the feature list, the round trip time (RTT) of a TCP flow is

Table 1 Features extracted from network traffic

Description	Count
number of packets and size of uploaded traffic	2
number of packets and size of downloaded traffic	2
number of downloaded and uploaded packets carrying payload	2
first byte of first packet of upstream and downstream traffic	2
min, max, mean, and var of uploaded traffic size	4
min, max, mean, and var of downloaded traffic size	4
min, max, mean, and var of uploaded inter-packet time	4
min, max, mean, and var of downloaded inter-packet time	4
min, max, mean, and var of RTT	4

Table 2 Ten more frequent EMs detected by HBN-SCA

Name	Count	Purity	Name	Count	Purity, %
DNS	241	74%	IMAP*	112	91
NetBIOS	93	89%	NTP	88	94
SAMBA	55	97%	SOCKSv5	54	95
ICMP*	52	99%	SSH	49	49
HTTP	35	90%	RTMP	32	86

* ICMP: Internet control message protocol; IMAP: Internet message access protocol

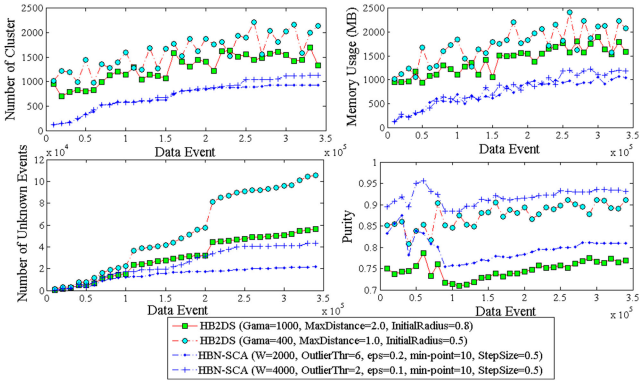

Fig. 7 *HBN-SCA against HB²DS output as stream proceeds*

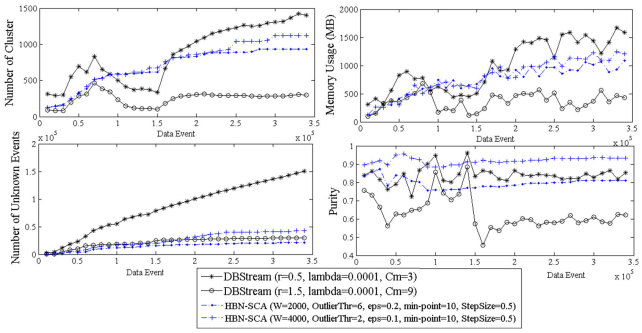

Fig. 8 *HBN-SCA against DBStream output as stream proceeds*

calculated as the total time between a sender transmitting a packet and the reception of its corresponding ack packet.

In the pilot study, default values for HBN-SCA are $W = 2000$, OutlierThr = 6, eps = 0.2, min-Point = 10, and StepSize = 0.5. HBN-SCA clusters network traffic into about 1000 EMs. An EM label is set as the majority label of its flows detected by nDPI. Accordingly, the EMs detected by HBN-SCA are categorised in Table 2. The purity is calculated as the average of purity values over all EMs with a similar label.

Fig. 7 shows HBN-SCA and HB²DS outputs as the stream proceeds. While HB²DS generates new clusters when there is no cluster to cover incoming flows, HBN-SCA runs DBScan algorithm over the window and generates new clusters regarding the results. This main difference, in addition to the forgetting mechanism, makes HBN-SCA creates more accurate clusters. With more accurate clusters, as Fig. 7 represents, HBN-SCA processes the input flow with less number of clusters than HB²DS. Moreover, the window model and the forgetting mechanism make HBN-SCA has less unknown flows compared to the number of unknown flows in HB²DS. In average, the purity of clusters in HBN-SCA is 5% over the purity of clusters in HB²DS. So, HBN-SCA not only decreases the number of unknown flows but also creates better clusters.

Fig. 8 compares HBN-SCA against DBStream outputs. DBStream outputs highly depend on the initial parameters (refer to Appendix for more results). Two of the best results are chosen to compare with HBN-SCA. In DBStream, when $r = 0.5$, the algorithm creates more clusters but the number of unknown flows increases. The main reason is that the clusters radius, r, is low and the algorithm does not update the radius according to the input flows. When we increase the radius r to 1, the algorithm creates fewer clusters and labels fewer flows as unknown, but the purity decreased significantly. The number of DBStream detected clusters is less than the number of HBN-SCA detected clusters, while the average purity of clusters detected by HBN-SCA is more than the purity of clusters detected by DBStream. The performance of DBStream is highly related to its initial parameters, because it cannot adapt itself with the input flows, while HBN-SCA window processing mechanism, which selects the initial cluster parameters regarding the input flows status, makes the clustering algorithm creates more accurate clusters.

Fig. 9 shows the results of DenStream and HBN-SCA. DenStream is a density-based clustering algorithm, which can find non-convex clusters. The number of DenStream detected clusters is less than the number of clusters detected by HBN-SCA. Since it cannot find clusters for most sessions and labelled them as unknown. When DenStream initial parameter epsilon (ε) is set to 0.1, the purity of clusters is similar to the purity of HBN-SCA detected clusters but the number of unknown sessions is significantly high. Moreover, the purity in HBN-SCA is more consistent than the purity in DenStream as stream proceeds.

4.4 Sensitivity analysis

The sensitivity of HBN-SCA results to its initial parameters is depicted through Fig. 10. Initial setup of HBN-SCA is $W = 2000$, OutlierThr = 6, eps = 0.2, min-Point = 10, and StepSize = 0.5. Then, the effects of W, OutlierThr, eps, min-Point, and StepSize parameters are measured on HBN-SCA's outputs.

The number of unknown flows varying different initial parameters is depicted in Fig. 10*a*. As the figure shows, the number of unknown flows changes from 1×10^4 to 2.2×10^4 when the

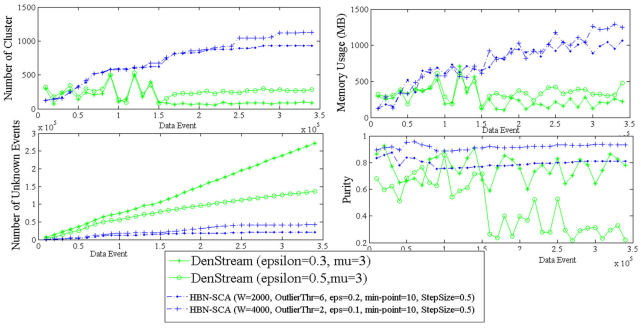

Fig. 9 *HBN-SCA against DenStream output as stream proceeds*

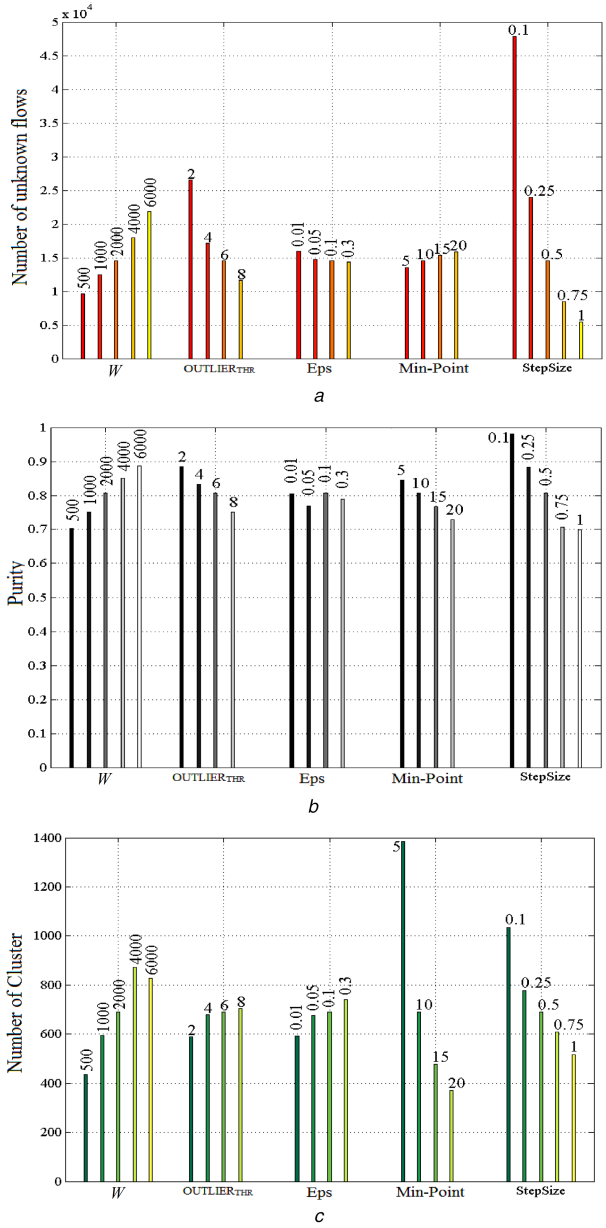

Fig. 10 *Sensitivity analysis of HBN-SCA varying different initial parameters*

(a) Number of unknown flows labeled by HBN-SCA varying different initial values, ***(b)*** Purity of clusters detected by HBN-SCA varying different initial values, ***(c)*** Number of clusters detected by HBN-SCA varying different initial values

widow size *W* alters from 500 to 6000. So, the window size has a positive correlation with the number of unknown flows. The effects of eps and min-Point on the number are low. However, the OutlierThr and StepSize parameters have negative correlations with the number of unknown flows. As these parameters increase, the algorithm tries to define new EMs more carefully, which increases the number of unknown flows.

The purity of HBN-SCA results varying different initial parameters is demonstrated in Fig. 10*b*. *W*, OutlierThr, and StepSize parameters have more effects on the purity of output clusters. When step Size = 0.1, HBN-SCA increases the EM radius a little, which increases the processing overhead and also increases the purity of detected clusters. There is no specific relation between eps and the purity values.

Fig. 10*c* presents the number of clusters varying different initial parameters. As the figure shows, min-Point and StepSize have a negative correlation with the number of clusters, while min-Point has more effect on the number of clusters. When min-Point = 5, the algorithm detects more than 1300 clusters. In this state, the purity is increased a little (Fig. 10*b*), which means when min-Point is set to 5 the algorithm does not detect accurate clusters. Considering $stepSize = 0.1$, the detected clusters number is increased while the purity is almost 98% where the algorithm detects precise clusters.

Table 3 Network analysing approaches and traffic dynamic behaviour issues

Name	Incremental	Outlier	Forgetting	Inline
Arora *et al.* [23]	—	—	—	✓
Lu *et al.* [24]	—	—	—	✓
Wang *et al.* [21]	—	✓	—	—
Zhang *et al.* [4]	—	✓	—	—
Casas *et al.* [12]	—	✓	—	✓
Jin *et al.* [13]	—	✓	—	✓
HB^2DS [14]	✓	✓	—	✓
HBN-SCA	✓	✓	✓	✓

4.5 Comparing high-bandwidth network analysing approaches

Table 3 summarises how the current network analysing approaches address the high-bandwidth network traffic dynamic behaviour issues including incremental learning, outlier handling, forgetting mechanism, and inline processing. As the table shows, the state-of-the-art in network traffic analysis did not consider the dynamic nature of high-bandwidth networks. Some of them have assumptions, including processing of each flow more than once, do not forget the expired information or processing an outlier free stream, which is impracticable in real networks. Utilising stream clustering concepts in HBN-SCA makes it more appropriate for dynamic behaviours of high-bandwidth networks.

Arora *et al*. [23] apply different machine-learning algorithms such as K-nearest neighbour, naive Bayes, support vector machine, and random forest to categorise network nodes on Netflow traffic. The approach reaches high performance in the categorising, but the aforementioned algorithms are not adaptive while they do not incrementally learn the flows and do not have forgetting mechanism and inline processing. Focusing on some popular network protocols such as FTP and HTTP, Lu *et al*. [24] use message size sequence and distribution to classify the protocols. While this approach does not address the adaptive learning requirements, it also lacks comprehensiveness issue to consider network administrators' issues.

Zhang *et al*. [4] and Wang *et al*. [21] try to distinguish unknown applications among high-bandwidth network traffics. These approaches obtain high performance and accuracy in detecting unknown applications and outliers, but they do not cluster the network traffic in more applications, which are required by network administrators.

Utilising ensemble learning methods, Casas *et al*. [12] and Jin *et al*. [13] propose inline network traffic classification approaches. Although these approaches achieve high accuracy and performance in analysing real-time network traffic, they are not adapted with the network traffic changes. So, as the traffic behaviours are changed or some unknown behaviours have appeared, the approaches do not support incremental learning and require relearning which is not applicable in high-bandwidth networks.

5 Conclusion and future works

Clustering high-bandwidth network traffics is required by real-time services such as detecting intrusions or monitoring end-user behaviours. Considering the traffic as a dynamic environment having a potentially infinite stream of flows, this study presented HBN-SCA as a window-based stream clustering algorithm to tackle high-bandwidth networks analysing challenges such as incremental learning, outlier handling, forgetting mechanism, and inline processing. Under the sliding window model, the algorithm maintains statistics and information about the recent traffic in an abstract data structure called EM. The EMs are used to increase the clustering accuracy and robustness against traffic outliers. Experimental results have empirically demonstrated that developing HBN-SCA yields high accuracy and efficiency in real network traffic and workloads.

Further researches can continue in several directions. First, the buffering mechanism of HBN-SCA is based on centre-point which imposed computation overhead. Decreasing the overhead should be studied in the network traffic analysing context. Second, the evaluation of HBN-SCA represented that as stream proceeds, the algorithm reaches a steady state where the frequency of changes is low. In this state, the size of the EMs set is high and finding the closest flow is a major part of the flow processing. The algorithm could find the distance between the flow and extended-meta-flows in the set using approximate nearest neighbour approaches such as [39]. The study of approximate nearest neighbour approaches and their effects on the performance and accuracy of clustering could be considered as a feature work.

Utilising DPI techniques, HBN-SCA could adaptively assign applications for each detected cluster. While DPI only processes the small amount of traffics, the algorithm tries to predict unknown flows' applications according to their clusters. This could decrease the overhead of DPI without the loss of accuracy, and could be considered as an interesting area for future work. We aim to investigate different ways in which live clustering information such as top fastest growing clusters can be represented. Monitoring the size of EMs, HBN-SCA could follow network resources usages such as a server or a website with no extra overhead. As further research, we plan to develop a precise and efficient distributed denial of service detection approach based on the monitoring capabilities of HBN-SCA.

6 References

[1] Moore, A.W., Papagiannaki, K.: 'Toward the accurate identification of network applications', in Dovrolis, C. (Ed.): Passive and Active Network Measurement (PAM 2005), Boston, MA, USA, 2005 (LNCS, **3431**), pp. 41-54

[2] Madhukar, A., Williamson, C.: 'A longitudinal study of P2P traffic classification'. 14th IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006 (MASCOTS 2006), Washington DC, USA, 2006, pp. 179–1881

[3] Alcock, S., Möller, J.P., Nelson, R.: 'Sneaking past the firewall: quantifying the unexpected traffic on major TCP and UDP ports'. Proc. 2016 ACM on Internet Measurement Conf., New York, USA, 2016, pp. 231–237

[4] Zhang, J., Chen, X., Xiang, Y., *et al.*: 'Robust network traffic classification', *IEEE/ACM Tran. Netw. (TON)*, 2015, **23**, pp. 1257–1270

[5] Perera, P., Tian, Y.C., Fidge, C., *et al.*: 'A comparison of supervised machine learning algorithms for classification of communications network traffic'. Int. Conf. on Neural Information Processing, Cham, Switzerland, 2017, pp. 445–454

[6] Manvi, S.S., Shyam, G.K.: 'Resource management for infrastructure as a service (IAAS) in cloud computing: a survey', *J. Netw. Comput. Appl.*, 2014, **41**, pp. 424–440

[7] Pathan, A.S.K.: '*The state of the art in intrusion prevention and detection*' (CRC Press, Hoboken, NJ, USA, 2014)

[8] Casas, P., Mazel, J., Owezarski, P.: 'Knowledge-independent traffic monitoring: unsupervised detection of network attacks', *IEEE Netw.*, 2012, **26**, pp. 13–21

[9] Dainotti, A., Pescape, A., Claffy, K.C.: 'Issues and future directions in traffic classification', *IEEE Netw.*, 2012, **26**, pp. 35–40

[10] Gharaee, H., Hosseinvand, H.: 'A new feature selection IDS based on genetic algorithm and SVM'. IEEE 2016 8th Int. Symp. on Telecommunications (IST), Tehran, Iran, 2016, pp. 139–144

[11] Tongaonkar, A., Torres, R., Iliofotou, M., *et al.*: 'Towards self adaptive network traffic classification', *Comput. Commun.*, 2015, **56**, pp. 35–46

[12] Casas, P., Mazel, J., Owezarski, P.: 'UNADA: unsupervised network anomaly detection using sub-space outliers ranking'. Int. Conf. on Research in Networking, Valencia, Spain, 2011, pp. 40–51

[13] Jin, Y., Duffield, N., Erman, J., *et al.*: 'A modular machine learning system for flow-level traffic classification in large networks', *ACM Trans. Knowl. Discov. Data*, 2012, **6**, pp. 4:1–4:34

[14] Noferesti, M., Jalili, R.: 'HB^{2}DS: a behavior-driven high-bandwidth network mining system', *J. Syst. Softw.*, 2017, **127**, pp. 266–277

[15] Aggarwal, C.C.: '*Data mining: the textbook*' (Springer, Switzerland, 2015)

[16] Gama, J.: '*Knowledge discovery from data streams*' (CRC Press, New York, USA, 2010)

[17] Silva, J.A., Faria, E.R., Barros, R.C., *et al.*: 'Data stream clustering: a survey', *ACM Comput. Surv.*, 2013, **46**, pp. 13:1–13:31

[18] Zarrabi-Zadeh, H., Mukhopadhyay, A.: 'Streaming 1-center with outliers in high dimensions'. Canadian Conf. on Computational Geometry (CCCG), Vancouver, Canada, 2009, pp. 83–86

[19] Ester, M., Kriegel, H.P., Sander, J., *et al.*: '*A density-based algorithm for discovering clusters in large spatial databases with noise*' (AAAI Press, Portland, OR, USA, 1996), pp. 226–231

[20] Bennett, M.A., Piggott, A.C., Garfield, D.J.M., *et al.*: 'Real-time network monitoring and security'. US Patent 9,769,276, 2017

[21] Wang, B., Zhang, J., Zhang, Z., *et al.*: 'Robust traffic classification with mislabelled training samples'. 2015 IEEE 21st Int. Conf. on Parallel and Distributed Systems (ICPADS), Melbourne, Australia, 2015, pp. 328–335

[22] Lin, R., Li, O., Li, Q., *et al.*: 'Unknown network protocol classification method based on semi-supervised learning'. 2015 IEEE Int. Conf. on Computer and Communications (ICCC), Chengdu, China, 2015, pp. 300–308

[23] Arora, D., Li, K.F., Loffler, A.: 'Big data analytics for classification of network enabled devices'. 2016 30th Int. Conf. on Advanced Information Networking and Applications Workshops (WAINA), Crans-Montana, Switzerland, 2016, pp. 708–713

[24] Lu, C.N., Huang, C.Y., Lin, Y.D., *et al.*: 'High performance traffic classification based on message size sequence and distribution', *J. Netw. Comput. Appl.*, 2016, **76**, pp. 60–74

[25] Gomes, H.M., Barddal, J.P., Enembreck, F., *et al.*: 'A survey on ensemble learning for data stream classification', *ACM Comput. Surv. (CSUR)*, 2017, **50**, p. 23

[26] Aggarwal, C.C., Han, J., Wang, J., *et al.*: 'A framework for clustering evolving data streams'. Proc. 29th Int. Conf. on Very Large Data Bases, VLDB Endowment, Berlin, Germany, 2003, vol. 29, pp. 81–92

[27] Garofalakis, M., Gehrke, J., Rastogi, R.: '*Data stream management: processing high-speed data streams*' (Springer, Berlin, Germany, 2016)

[28] Amini, A., Ying, W.: 'Dengris-stream: a density-grid based clustering algorithm for evolving data streams over sliding window'. Proc. Int. Conf. on Data Mining and Computer Engineering, 2012, pp. 206–210

[29] Chen, Y., Tu, L.: 'Density-based clustering for real-time stream data'. Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, New York, NY, USA, 2007, pp. 133–142

[30] Aggarwal, C.C., Han, J., Wang, J., *et al.*: 'A framework for projected clustering of high dimensional data streams'. Proc. 30th Int. Conf. on Very Large Data Bases, Toronto, Canada, 2004, vol. 30, pp. 852–863

[31] Ren, J., Ma, R.: 'Density-based data streams clustering over sliding windows'. 2009 Sixth Int. Conf. on Fuzzy Systems and Knowledge Discovery, Tianjin, China, 2009, pp. 248–252

[32] Hahsler, M., Bolaños, M.: 'Clustering data streams based on shared density between micro-clusters', *IEEE Trans. Knowl. Data Eng.*, 2016, **28**, pp. 1449–1461

[33] Cao, F., Estert, M., Qian, W., *et al.*: 'Density-based clustering over an evolving data stream with noise'. Proc. 2006 SIAM Int. Conf. on Data Mining, Bethesda, MD, USA, 2006, pp. 328–339

[34] PF_RING: High-speed packet capture, filtering and analysis. URL: Available at https://www.ntop.org/products/packet-capture/pf_ring, 2018, accessed: 2018-05-11

[35] Hahsler, M., Bolanos, M., Forrest, J.: 'Introduction to stream: an extensible framework for data stream clustering research with *r*', *J. Stat. Softw.*, 2017, **76**, pp. 1–50

[36] Bifet, A., Holmes, G., Kirkby, R., *et al.*: 'MOA: massive online analysis', *J. Mach. Learn. Res.*, 2010, **11**, pp. 1601–1604

[37] Deri, L., Martinelli, M., Bujlow, T., *et al.*: 'nDPI: open-source high-speed deep packet inspection'. 2014 Int. Wireless Communications and Mobile Computing Conf. (IWCMC), Nicosia, Cyprus, 2014, pp. 617–622

[38] Wang, B., Zhang, J., Zhang, Z., *et al.*: 'Traffic identification in big internet data', in Shui, Y., Guo, S. (Eds.): '*Big data concepts, theories, and applications*' (Springer, Cham, Switzerland, 2016), pp. 129–156

[39] Muja, M., Lowe, D.G.: 'Scalable nearest neighbor algorithms for high dimensional data', *IEEE Trans. Pattern Anal. Mach. Intell.*, 2014, **36**, pp. 2227–2240

7 Appendix: Evaluation result details

The details of evaluation measures on clusters detected by the stream clustering algorithms are represented in Table 4. The selected measures are the number of clusters, the number of unknown sessions, SSE, silhouette, purity, and rand index. In the table, the average of the evaluating measures during the pilot study is shown.

Table 4 Cluster's characterisations detected by stream clustering algorithms with different initial parameters on the pilot study network traffic

Algorithm	#Clusters	#Unknown	SSE	Silhouette	Purity	Rand
DBStream (r = 0.5, λ = 0.01, gaptime = 1000 L, Cm = 3, α = 0.1, minweight = 0)	8.03	296,765	223,903.04	−0.223	0.868	0.228
DBStream (r = 0.5, λ = 0.001, gaptime = 1000 L, Cm = 3, α = 0.1, minweight = 0)	220.80	155,583	97,475.96	−0.124	0.886	0.615
DBStream (r = 0.5, λ = 0.0001, gaptime = 1000 L, Cm = 3, α = 0.1, minweight = 0)	865.74	92,828	63,856.62	−0.018	0.876	0.730
DBStream (r = 0.7, λ = 0.001, gaptime = 1000 L, Cm = 3, α = 0.1, minweight = 0)	202.44	112,023	91,185.32	0.006	0.802	0.695
DBStream (r = 0.7, λ = 0.001, gaptime = 1000 L, Cm = 9, α = 0.1, minweight = 0)	106.38	150,013	117,102.70	−0.064	0.839	0.634
DBStream (r = 0.7, λ = 0.001, gaptime = 10000 L, Cm = 3, α = 0.1, minweight = 0)	232.18	110,430	90,078.90	−0.002	0.801	0.698
DBStream (r = 0.9, λ = 0.0001, gaptime = 1000 L, Cm = 3, α = 0.1, minweight = 0)	573.59	54,713	54,511.28	0.105	0.777	0.784
DBStream (r = 1.5, λ = 0.0001, gaptime = 1000 L, Cm = 3, α = 0.1, minweight = 0)	409.18	33,094	49,523.99	0.196	0.714	0.812
DBStream (r = 1.5, λ = 0.0001, gaptime = 1000 L, Cm = 9, α = 0.1, minweight = 0)	249.50	39,451	61,217.23	0.186	0.675	0.806
D-Stream (grid size = 10, λ = 0.001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 3)	1337.35	22,553	5,377,983.79	−0.083	0.795	0.838
D-Stream (grid size = 5, λ = 0.001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 3)	1822	24,749	1,103,009.92	−0.076	0.806	0.836
D-Stream (grid size = 5, λ = 0.0001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 3)	5234.94	19,425	1,096,908.82	−0.074	0.817	0.839
D-Stream (grid size = 1, λ = 0.001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 9)	9691.09	52,467	24,267.58	−0.072	0.814	0.786
D-Stream (grid size = 10, λ = 0.0001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 3)	3800.56	15,769	5,367,457.79	−0.083	0.809	0.841
D-Stream (grid size = 100, λ = 0.0001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 3)	2761.38	14,976	659,093,443	−0.086	0.806	0.841
d-Stream (grid size = 50, λ = 0.0001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 3)	2892.85	14,979	160,923,116	−0.086	0.806	0.841
d-Stream (grid size = 20, λ = 0.0001, gaptime = 1000 L, Cm = 3, Cl = 0.1, ε = 3)	3356.59	15,134	24,012,365	−0.085	0.807	0.841

Algorithm	#Clusters	#Unknown	SSE	Silhouette	Purity	Rand
CluStream (m = 50, horizon = 1000, t = 2)	50	324,862	77,414.86	NA	0.717	0.256
CluStream (m = 500, horizon = 1000, t = 2)	500	339,697	56,969.41	NA	0.798	0.221
CluStream (m = 500, horizon = 100, t = 10)	500	339,918	86,550.39	NA	0.598	0.220
CluStream (m = 1000, horizon = 1000, t = 10)	1000	339,867	51,635.99	NA	0.749	0.221
CluStream (m = 5000, horizon = 100, t = 10)	5000	339,856	41,689.34	NA	0.603	0.220
DenStream (ε = 0.3, μ = 3, β = 0.6, λ = 0.001, initPoint = 100, offline = 2)	20.59	274,751	200,412.33	−0.122	0.735	0.359
DenStream (ε = 0.1, μ = 3, β = 0.1, λ = 0.001, initPoint = 100, offline = 2)	168.97	293,848	115,316.74	−0.302	0.850	0.302
DenStream (ε = 0.1, μ = 3, β = 0.2, λ = 0.001, initPoint = 1000, offline = 2)	123.74	298,911	129,704.39	−0.259	0.847	0.285
DenStream (ε = 0.1, μ = 1, β = 0.2, λ = 0.001, initPoint = 2000, offline = 2)	123.76	298,801	129,711.79	−0.259	0.847	0.285
DenStream (ε = 0.5, μ = 3, β = 0.2, λ = 0.001, initPoint = 100, offline = 2)	241.62	141,347	98,534.85	0.012	0.573	0.668
DenStream (ε = 0.5, μ = 3, β = 0.2, λ = 0.001, initPoint = 1000, offline = 2)	257.71	138,315	98,080.54	0.018	0.562	0.671
DenStream (ε = 0.5, μ = 3, β = 0.2, λ = 0.001, initPoint = 500, offline = 6)	264.09	136,406	167,635.29	0.166	0.468	0.557
DenStream (ε = 0.5, μ = 3, β = 0.2, λ = 0.001, initPoint = 500, offline = 10)	264.09	136,406	207,863.69	0.189	0.416	0.534
DenStream (ε = 0.1, μ = 1, β = 0.2, λ = 0.001, initPoint = 100, offline = 2)	204.5	293,700	110,179.476	−0.317	0.856	0.301
DenStream (ε = 0.3, μ = 3, β = 0.2, λ = 0.001, initPoint = 100, offline = 2)	25.06	271,666	169,086.81	−0.182	0.761	0.358
DenStream (ε = 0.3, μ = 3, β = 0.4, λ = 0.001, initPoint = 100, offline = 2)	25.59	276,237	187,351.68	−0.178	0.736	0.344
DenStream (ε = 0.1, μ = 1, β = 0.2, λ = 0.001, initPoint = 4000, offline = 2)	121.82	297,677	129,591.24	−0.251	0.845	0.292
HB^{2}DS (Γ = 1000, maxDistance = 2.0, inititlaRadius = 0.8)	950.38	31,467	453.22	0.134	0.748	0.488
HB^{2}DS (Γ = 1000, maxDistance = 4.0, inititlaRadius = 0.8)	942.55	25,639	470.87	0.132	0.734	0.536
HB^{2}DS (Γ = 400, maxDistance = 1.0, inititlaRadius = 0.5)	967.02	52,062	304.28	0.217	0.859	0.221
HB^{2}DS (Γ = 2000, maxDistance = 6.0, inititlaRadius = 1.0)	920.94	19,034	490.45	0.123	0.437	0.553
HB^{2}DS (Γ = 2000, maxDistance = 8.0, inititlaRadius = 1.2)	900.12	12,003	536.97	0.128	0.325	0.573
HB^{2}DS (Γ = 800, maxDistance = 2.0, inititlaRadius = 0.5)	963.82	40,317	342.35	0.182	0.836	0.349
HB^{2}DS (Γ = 1000, maxDistance = 4.0, inititlaRadius = 1.0)	926.17	22,781	477.04	0.142	0.692	0.540

Algorithm	#Clusters	#Unknown	SSE	Silhouette	Purity	Rand
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.5)	689.02	19,355	247.78	0.187	0.806	0.724
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 2, eps = 0.1, min-Point = 10, StepSize = 0.5)	589.21	40,047	91.437	0.147	0.885	0.643
HBN-SCA (WindowSize = 6000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.5)	827.68	35,317	117.243	0.180	0.887	0.702
HBN-SCA (WindowSize = 4000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.5)	873.18	25,605	184.813	0.164	0.850	0.686
HBN-SCA (WindowSize = 1000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.5)	595.76	16,704	327.07	0.114	0.751	0.728
HBN-SCA (WindowSize = 500, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.5)	436.29	14,143	408.881	0.141	0.704	0.749
HBN-SCA (WindowSize = 500, OUTLIERTHR = 2, eps = 0.1, min-Point = 10, StepSize = 0.5)	514.91	23,614	236.067	0.135	0.774	0.584
HBN-SCA (WindowSize = 500, OUTLIERTHR = 4, eps = 0.1, min-Point = 10, StepSize = 0.5)	485.53	16,806	339.83	0.153	0.712	0.669
HBN-SCA (WindowSize = 1000, OUTLIERTHR = 2, eps = 0.1, min-Point = 10, StepSize = 0.5)	565.83	27,561	163.145	0.146	0.837	0.627
HBN-SCA (WindowSize = 1000, OUTLIERTHR = 4, eps = 0.1, min-Point = 10, StepSize = 0.5)	589.15	20,239	254.69	0.117	0.793	0.658

Algorithm	#Clusters	#Unknown	SSE	Silhouette	Purity	Rand
HBN-SCA (WindowSize = 500, OUTLIERTHR = 8, eps = 0.1, min-Point = 10, StepSize = 0.5)	374.53	12,858	402.624	0.131	0.655	0.728
HBN-SCA (WindowSize = 1000, OUTLIERTHR = 8, eps = 0.1, min-Point = 10, StepSize = 0.5)	589.14	20,239	254.69	0.128	0.793	0.692
HBN-SCA (WindowSize = 4000 ,OUTLIERTHR = 8, eps = 0.1, min-Point = 10, StepSize = 0.5)	893.85	19,197	246.74	0.164	0.828	0.649
HBN-SCA (WindowSize = 4000, OUTLIERTHR = 2, eps = 0.1, min-Point = 10, StepSize = 0.5)	745.18	47,077	88.36	0.156	0.928	0.595
HBN-SCA (WindowSize = 6000, OUTLIERTHR = 2, eps = 0.1, min-Point = 10, StepSize = 0.5)	745.18	47,077	88.36	0.193	0.928	0.483
HBN-SCA (WindowSize = 6000, OUTLIERTHR = 8, eps = 0.1, min-Point = 10, StepSize = 0.5)	873.09	19,340	191.62	0.181	0.840	0.703
HBN-SCA (WindowSize = 4000, OUTLIERTHR = 4, eps = 0.1, min-Point = 10, StepSize = 0.5)	779.91	32,832	116.19	0.174	0.899	0.609
HBN-SCA (WindowSize = 6000, OUTLIERTHR = 4, eps = 0.1, min-Point = 10, StepSize = 0.5)	806.82	27,201	128.73	0.180	0.894	0.667
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.3, min-Point = 10, StepSize = 0.5)	740.32	19,488	267.79	0.192	0.789	0.686
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.4, min-Point = 10, StepSize = 0.5)	781.12	16,433	336.61	0.177	0.778	0.692
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.05, min-Point = 10, StepSize = 0.5)	675.76	19,942	266.39	0.189	0.769	0.711

Algorithm	#Clusters	#Unknown	SSE	Silhouette	Purity	Rand
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.2, min-Point = 10, StepSize = 0.5)	692.26	19,338	259.37	0.171	0.785	0.717
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.01, min-Point = 10, StepSize = 0.5)	592.44	20,955	213.64	0.169	0.804	0.679
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 15, StepSize = 0.5)	476.82	20,639	192.12	0.157	0.686	0.554
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 5, StepSize = 0.5)	1384.21	18,458	412.12	0.164	0.764	0.547
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 20, StepSize = 0.5)	369.85	21,294	188.48	0.174	0.649	0.598
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 25, StepSize = 0.5)	294.59	21,118	154.95	0.172	0.640	0.572
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 30, StepSize = 0.5)	250.62	21,353	159.12	0.155	0.614	0.566
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.1)	1035.09	92,767	36.95	0.651	0.922	0.746
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.25)	778.26	36,761	129.93	0.315	0.804	0.618
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 0.75)	609.18	11,232	421.23	0.149	0.626	0.613
HBN-SCA (WindowSize = 2000, OUTLIERTHR = 6, eps = 0.1, min-Point = 10, StepSize = 1)	517.03	7881	511.16	0.133	0.618	0.551