Evolving Systems (2016) 7:129–143
DOI 10.1007/s12530-016-9152-x

ORIGINAL PAPER

Online network traffic classification with incremental learning

H. R. Loo[1] · **M. N. Marsono**[1]

Received: 31 August 2015 / Accepted: 3 April 2016 / Published online: 22 April 2016

Abstract Conventional network traffic detection methods based on data mining could not efficiently handle high throughput traffic with concept drift. Data stream mining techniques are able to classify evolving data streams although most techniques require completely labeled data. This paper proposes an improved data stream mining algorithm for online network traffic classification that is able to incrementally learn from both labeled and unlabeled flows. The algorithm uses the concept of incremental *k*-means and self-training semi-supervised method to continuously update the classification model after receiving new flow instances. The experimental results show that the proposed algorithm is able to classify 325 thousands flow instances per second and achieves up to 91–94 % average accuracy, even when using 10 % of labeled input flows. It is also able to maintain high accuracy even in the presence of concept drifts. Although there are drifts detected in the datasets evaluated using the Drift Detection Method, our proposed method with incremental learning is able to achieve up to 91–94 % accuracy compared to 60–69 % without incremental learning.

Keywords Incremental learning · Online classification · Network traffic classification · Data stream mining

✉ M. N. Marsono
nadzir@fke.utm.my

H. R. Loo
loohuiru@gmail.com

[1] Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Johor Bahru, Malaysia

1 Introduction

Classification of network traffic is a critical task in network processing for traffic management and also to improve quality-of-service. Network grows in terms of speed and size, and classification of network traffic becomes more challenging as traffic volume increases. Hence, it is expensive to store it partially or entirely for conventional supervised learning. Besides, non-traditional application traffic such as peer-to-peer, online gaming, and multimedia also complicate network traffic classification as labeling these traffic types is not trivial. Peer-to-peer is one of the examples of such applications that use dynamic port numbers and port masquerading that is difficult to be differentiated from background traffic.

Port-based traffic classification which is based on 5-tuple information can no longer provides accurate detection of Internet traffic (Dainotti et al. 2012). Meanwhile, payload-based classification is questionable due to privacy issue (Dainotti et al. 2012). Machine learning based traffic classification has been introduced to overcome the limitation of port-based and payload-based traffic classifier (Tian et al. 2008; Li 2012; Mingliang et al. 2009; Raahemi and Mumtaz 2008; Raahemi et al. 2008). However, conventional machine learning based traffic classification only performs one-time training which make them incapable to adapt to new concepts. In order to classify network traffic in todays dynamic environment, data stream mining algorithms (Abdulsalam 2008; Aggarwal et al. 2004; Domingos and Hulten 2000; Hulten et al. 2001; Street and Kim 2001; Wang et al. 2003) have been introduced to overcome the shortcoming of conventional data mining algorithms. These algorithms are designed to handle concept drift, unlearn outdated data and adapt to new network state.

Most data stream classification algorithms e.g. (Aggarwal et al. 2004; Domingos and Hulten 2000) assume that all incoming data streams are fully labeled. However, in the context of online network traffic classification, labeling all incoming flows are not feasible as flow labeling is time consuming and it could not be done accurately without human inputs. Thus, an online network traffic classifier must be able to learn from both labeled and unlabeled flows instance. In order to classify traffic online, traffic features must be acquired on-the-fly for processing in the network. Monemi et al. (2013) have also shown that the use of online features not only can achieve accuracy similar to full flow features, it also improves classification speed.

In this paper, we propose an online classification method which is aimed for online network traffic classification. By applying incremental *k*-means, the classification model can learn from unlabeled and labeled data. Hence, it can adapt to new concept and can reduce data labeling effort for learning. However, not all of the unlabeled data are informative and suitable to be used for learning. We propose a selection method that chooses only data with high prediction confidence based on criteria that can be obtained simultaneously during classification process. Besides, labeled data will also be handle using different handling method based on the confidence level. Reconstruction and reduction of clusters are done to make sure only useful and updated clusters are stored in the classification model. Our proposed method can classify online network traffic based on statistical flow features which can be acquire on-the-fly to enable online classification. The proposed method is able to maintain high accuracy even in the presence of concept drift.

The remainder of this paper is structured as follows. Section 2 introduces related works on incremental learning for network traffic classification and semi-supervised incremental learning. Section 3 explains our proposed method and Sect. 4 analyzes the experimental results. We conclude our paper in Sect. 5.

2 Related work

Data stream mining techniques have been proposed to perform online classification and one-pass learning. The learning mechanism can be divided into two main methods, incremental learning and retraining. Incremental learning algorithms such as Very Fast Decision Tree (VFDT) (Domingos and Hulten 2000) and its variation Concept-Adaptive VFDT (Hulten et al. 2001) are widely used in online traffic classification (Tian et al. 2008; Raahemi et al. 2008; Mingliang et al. 2009) as these algorithms have moderate complexity and they can naturally adapt to new concepts. However, most incremental learning algorithm suffer from low accuracy as the classification model is built based on entire data stream history (Aggarwal et al. 2006), which may consist of concepts that are no longer true. The impact of such issue can be alleviated by proposing unlearning the classification model. Angelov and Zhou (2008), Lughofer and Angelov (2009, 2011) proposed online classification based on evolving fuzzy system which are able to perform incremental learning through evolution when drifts are detected. The classification rate of such system are comparable to well established offline incremental classifier such as incremental C4.5 and incremental kNN.

Retraining does not update classification model while classifying. It can be done either using a drift detector such as Drift Detection Method (Gama et al. 2004) or sliding window to initiate retraining of a model. This method is implemented in most traffic classifiers such as (Li and Moore 2007; de Souza et al. 2014). However, most state-of-the-art drift detectors (Gama et al. 2004; Baena-García et al. 2006; Minku and Yao 2012) are based on accuracy feedback, which are not feasible when the number of flow labels is limited.

Although the proposes of network traffic classifiers using data stream mining techniques seem to be more common, most of the proposed methods still rely on fully labeled data. Only a few of such research works (Shrivastav and Tiwari 2010; Qian et al. 2008; Erman et al. 2007) show the use of semi-supervised concept in traffic classification to reduce the data labeling effort. Shrivastav and Tiwari (2010) has proposed a semi-supervised algorithm to built *k*-means clustering model using both labeled and unlabeled data. The proposed algorithm uses a batch learning approach that does not include online classification and incremental learning capability. Qian et al. (2008) developed a traffic classifier using the Gaussian Mixture Model to learn from labeled and unlabeled network flows. This approach does not fulfill data stream classification requirement as it does not perform online classification and it does not have a mechanism to update and adapt the classification model to concept changes. Erman et al. (2007) shows its ability to perform classification in either offline or real-time models. However, significant drop in accuracy was reported when the proposed work performs classification in real-time.

Masud et al. (2012) and Liu et al. (2014) proposed semi-supervised ensembles learning with label propagation method called ReaSC and ECM-BDF, respectively. Both algorithms use batch learning method to train new data and to update ensemble model. These algorithms allow new concepts to be learned without forgetting previously known concepts. However, the time for retraining is highly dependent on the batch size (also known as chunk size). Big batch size results in slow learning whereas small chunk

size will reduce the reliability of the training model. Semi-supervised in incremental learning had been proposed by Bertini Jr et al. (2012), Loo et al. (2014). Bertini Jr et al. (2012) extend k-associated graph with graph-based semi-supervised learning and incremental learning (KAOGINC-SSL) and analyze on synthetic and real datasets. This work shows the implementation of semi-supervised method in incremental learning for data stream classification purpose. However, this method shows lower accuracy ($\leq$80 %) in real dataset for 10 % labeled data. On the other hand, Loo et al. (2014) proposed an online data stream learning and classification with limited labels algorithm based on incremental *k*-means and self-training method. Loo et al. (2014) have chosen self-training method that is the simplest semi-supervised method and include a selection step to choose only reliable unlabeled data to train, the work is able to achieve high accuracy ($\leq$90 %) even with only 1 % labeled data. Loo and Marsono (2015) extended the prior work in (Loo et al. 2014) by proposing partitioning of classification model to increase classification speed. Both works assume that labeling of instances can be done immediately after the learning process. However this is not applicable in network traffic classification as flow labeling involve manual steps. In order to apply the algorithm on network traffic classification, certain modification are needed such as the learning process for labeled flows and selecting the flow features that can be extracted online.

3 Overview of proposed method

Our proposed self-adaptive online traffic classification model consists of two main processes: classification and learning. Some of the terms and terminology used in this paper are as follows:

1. *Flow:* The network traffic that belongs to a process-to-process communication.
2. *Instance:* A tupple of attributes.
3. *Flow features:* The attributes or statistical features extracted from a flow, e.g. number of bytes in payload.
4. *Flow instance:* The instance made up of flow features which represent a flow.

The classification process performs online classification on network traffic by using online flow features, while the learning process performs incremental learning to update the classification model. Both processes will be discussed in details in Sects. 3.2 and 3.4, respectively. Figure 1 shows the overview of the proposed method. The selection module performs the selection of flow instances to be learned to avoid mislearning (see Sect. 3.3). Manual labeling is not covered in this paper. An example of such technique is the groundtruth method which is discussed in Gringoli et al. (2009).

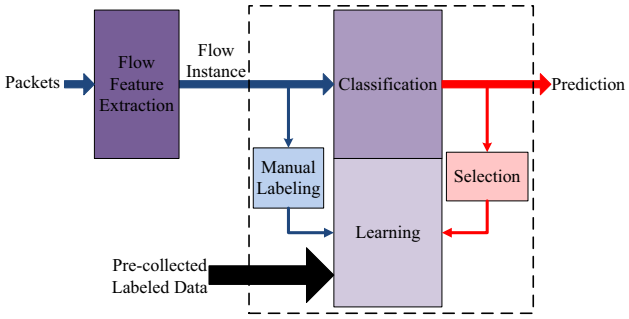

Fig. 1 Overview of proposed method

3.1 Classification model initialization

Before online classification can be performed, the classification model need to be initialized. This process is performed once during start up to prepare the *base* classifier model. In this stage, the supervised *k*-means technique is used to cluster the collected batch labeled flow instances into k_d clusters, where k_d is the desired number of clusters for the model. In order to increase classification efficiency, the classification model M is made up from B smaller micro-models that are located in different ranges in the Euclidean space. To perform this, the pre-collected flow instances are distributed to B chunks according to the distance to origin D_o [Algorithm (1), line 8–10], where the distance $D_o(\mathbf{x_i})$ is defined as

$$D_o(\mathbf{x_i}) = \sqrt{\sum_{m=0}^{d} (x_i^m)^2} \tag{1}$$

where $\mathbf{x_i}$ is a flow instance with d flow features. Each micro-model is then built using respective chunk of flow instances [Algorithm (1), line 11–19].

The initialization of centroids is based on the method suggested in (Masud et al. 2012) so that the initial class of clusters are proportional to the collected data class distribution. The clusters are then compressed to sufficient statistics known as Clustering Features (CF). CF as proposed in (Zhang et al. 1996) is a 3-tupple information that summarizes information about a cluster. Given N_j d-dimensional data points ($\mathbf{x_i}$) in a cluster j, where $i = 1, 2, \ldots, N_j$, the CF of the cluster is defined as 3 tupple $CF_j =< N_j, \mathbf{LS_j}, \mathbf{SS_j} >$, where

$$\mathbf{LS_j} = \sum_{i=1}^{N_j} \mathbf{x_i} \tag{2}$$

$$\mathbf{SS_j} = \sum_{i=1}^{N_j} (\mathbf{x_i})^2 \quad (3)$$

In our classification model, we modify and add the timestamp T_j and class information y_j to represent the cluster $CF_j =< N_j, \mathbf{LS_j}, \mathbf{SS_j}, T_j, y_j >$. Merging new ($\mathbf{x_i}$) to cluster j is based on the Additivity Theorem (Zhang et al. 1996) with T_j and y_j remain unchanged, such that

$$CF_j =< (N_j + 1), (\mathbf{LS_j} + \mathbf{x_i}), (\mathbf{SS_j} + (\mathbf{x_i})^2), T_j, y_j > \quad (4)$$

Raw data are discarded in order to save memory space. The k_d clusters that are represented by k_d clustering features are used for classification and the clusters may be modified based on a newly received flow instance. Algorithm 1 shows the overall steps of classification model initialization. During model initialization, pre-collected flow instances are divided into B sets, and the supervised k-means method is used to create a micro-model for each set of flow instances. Created clusters are summarized as Clustering Features CF with timestamp 0.

Algorithm 1 Classification model initialization

```
x_i: Pre-collected flow instances (array of online features)
y_i: True labels for x_i
D_o: Distance of x_i to origin
B: Number of micro-model
N: Number of pre-collected flow instances
k_d: Number of desired cluster
k_B: Number of cluster in a micro-model
for each x_i in N do
    D_o[i] ← D_o(x_i)
end for
Quicksort(D_o)
for every j micro-model do
    Border = D_o[N/B × (j + 1)]
    Assign x_i that smaller than border to j
    Assign centroid
    k-means x_i in j to k_B cluster
    Summarize k_B cluster into Clustering Feature, CF
    Store clusters in time-series and set timestamp to 0
end for
```

3.2 Online traffic classification

Classification starts upon receiving an incoming flow instance ($\mathbf{x_i}$). The distance to origin $D_o(\mathbf{x_i})$ is calculated to find the respective micro-model. In the micro-model, the distance between cluster's centroid ($\mu_\mathbf{j}$) and $\mathbf{x_i}$ is computed using Eq. (5). The nearest cluster (C_1) and second nearest cluster (C_2) are then determined. Assuming that the real class label of a flow instance is unknown (unlabeled flow instances), the predicted class $y'(\mathbf{x_i})$ will be classified to class of C_1 with respect to $\mathbf{x_i}$. Timestamp for C_1, T_{C_1}will then be increased by one.

$$D(\mathbf{x_i}, \mu_\mathbf{j}) = \sqrt{\sum_{m=0}^{d} (x_i^m - \mu_j^m)^2} \quad (5)$$

Table 1 Prediction confidence level

Confidence level	Conflicting neighbour ($y_{C_1} \neq y_{C_2}$)	In-boundary ($\mathbf{x_i} \in R_{C_1}$)
L_0	Yes	–
L_1	No	No
L_2	No	Yes

where $\mu_\mathbf{j}$ is the centroid of cluster j and $\mu_\mathbf{j} = \frac{\mathbf{LS_j}}{N_j}$.

3.3 Selection of learning instances

As we are applying self-training method in our learning algorithm, all predicted label are assumed to be accurate. While this assumption is not entirely true in incremental learning, certain threshold of false positive must be expected. Thus by having selection criteria to select only those flow instances with high confidence of prediction, false learning can be reduced. The selection criteria are designed such that they make use of the information from classification process that do not need extra computation. Extra computation will make the model consumes more time for learning and makes incremental learning inefficient.

Confidence level is divided into three levels $< L_0, L_1, L_2 >$ as shown in Table 1. The criteria are to determine confidence level in terms of label conflict within two nearest clusters (conflicting neighbor) and condition when an instance is within the nearest cluster's boundary (in-boundary). Conflicting neighbors are set to *true* when both nearest clusters belong to different classes or set to *false* when they belong to the same class.

The boundary of the nearest cluster C_1 is determined by the cluster's average radius R_{C_1} defined by Eq. (6) (Zhang et al. 1996). However, for clusters with $N = 1$, Eq. (6) is no longer valid as the subtraction in numerator will results in zero denominator.

$$R_{C_1} = \sqrt{\frac{\sum_{m=0}^{d} SS_{C_1}^m - \left(\frac{\sum_{m=0}^{d} (LS_{C_1}^m)^2}{N_{C_1}}\right)}{N_{C_1}}} \quad (6)$$

This consideration was not analyzed in (Loo et al. 2014). Aggarwal et al. (2004) calculate the maximum boundary based on the nearest neighbor's maximum boundary. However, the boundary of such clusters are not always similar, and to determine nearest neighbors will increase the system computation complexity by $O(kd)$. Thus we propose that the boundary of a cluster with only one instance can be determined by the similarity of attributes. Each attribute of a cluster instance is compared with the attributes of an incoming instance. An attribute is

Table 2 Injection handling method

Scenario	Confidence level	Prediction	Procedure
1	0	True	Merge $\mathbf{x_L}$ with C_1 if $\mathbf{x_L} \in R_{C_1}$, else add new cluster for $\mathbf{x_L}$
2	0	False	Merge $\mathbf{x_L}$ with C_2 if $\mathbf{x_L} \in R_{C_2}$ and $y_x = y_{C_2}$, else add new cluster for $\mathbf{x_L}$. Delete C_1
3	1	True	If $\mathbf{x_L}$ is not in $R(C_2)$, add new cluster for $\mathbf{x_L}$
4	1	False	Add new cluster for $\mathbf{x_L}$
5	2	True	Do nothing
6	2	False	Erase C_1 and add new cluster for $\mathbf{x_L}$

Algorithm 2 Semi supervised Incremental Learning

```
1: x_L2: L2 flow instance
2: C1: Nearest clusters from x_L2
3: while new x_L2 do
4:    merge x_L2 to C1
5: end while

6: —Injecting Labeled Instance (upon receiving)—
7: if receive (x_L, y_L) then
8:    classify x_L
9:    inject x_L based on the rules in Table 2
10: end if

11: —Micro-model Reconstruction (after one chunk)—
12: for all micro-model j do
13:    k_j: Number of clusters in micro-model j
14:    k_d: Desired number of clusters
15:    while k_j > k_d do
16:       delete clusters with timestamp 0
17:       reduce clusters timestamp by 1
18:    end while
19: end for
20: gather all clusters in classification model
21: reconstruct micro-model
```

considered similar to other attributes if the ratio between them is within 10 % of each other ($0.9 \geqslant \frac{x_1}{x_2} \geqslant 1.1$). A parameter boundary threshold is needed to define maximum non-similar attributes between instances and the cluster centroid that can be tolerated to include an incoming instance in the respective cluster. If the number of non-similar attributes are lower than the boundary threshold, it is considered to be within the boundary of the particular cluster.

The confidence level can be determined as follow: Let C_1 be the nearest cluster and C_2 be the second nearest cluster, y_{C_i} be the class of C_i, R_{C_i} be the radius of C_i, and $\mu_{\mathbf{C_i}}$ be the centroid of C_i. Confidence level is set to L_0 by default. In the case of $y_{C_1} = y_{C_2}$, confidence level will increase by one (L_1). If $\mathbf{x_i} \in R_{C_1}$ for C_1 with more than one instance or $\mathbf{x_i} \approx \mu_{\mathbf{C_1}}$ for C_1 with only one instance, confidence level will increase to two (L_2) if the condition $y_{C_1} = y_{C_2}$ is satisfied.

3.4 Semi-supervised incremental learning

Only flow instances with confidence level L_2 ($\mathbf{x_{L_2}}$) are used in incremental learning. As shown in Algorithm 2, (line 3–5), flow instances with confidence level L_2 will be merged with the nearest cluster C_1 based on Eq. (4). The learning will then update the classification model. Upon receiving labeled instances, injection process will be initiated. As the input flow instances are in streams, changes in distribution and concept are expected. Outdated knowledge need to be deleted and the micro-model need to be reconstructed based on recent clusters.

3.4.1 Injecting labeled instances

We assume that some flows will be labeled externally, and fed into the model once they are ready [Algorithm (2), line 6–11]. Thus, the labeled instances will be treated as a new flow instances and re-classified in order to get the necessary information such as confidence level and prediction results. In order to achieve minimum effort in injection, different handling methods based on the confidence level and prediction results are proposed for handling different possible scenarios. Table 2 shows the injection handling method for several scenarios. False prediction in trained instances will cause the trained cluster to be deleted immediately since the cluster is no longer reliable. Besides, new cluster with $\mathbf{x_L}$ will be added if $\mathbf{x_L}$ is not in the boundary of C_1. This is true except for scenarios 2 and 3, since there is a possibility of $\mathbf{x_L}$ to be within the boundary of C_2. Figure 2a illustrates the example of boundary condition when $\mathbf{x_L}$ is nearer to C_1 but is in the boundary of C_2. Merging $\mathbf{x_L}$ to C_2 will shift boundary of C_2 towards C_1 and can result in an overlap (Fig. 2b). Thus, when C_1 and C_2 are of different classes, one of these needs to be deleted. If they belong to the same class, $\mathbf{x_L}$ will be ignored since its learning brings no significant changes to the model.

3.4.2 Micro-model reconstruction

In order to prevent the storing of all outdated clusters that may result in imbalanced micro-model, a micro-model reconstruction process is performed after a user-predefined number of instances (chunks) have been received [Algorithm (2), line 11–21]. Clusters that are not utilized or under utilized will be deleted as they do not contribute to the classification decision. In addition, the reduction also aims to reduce memory footprint and classification time. This reduction process includes the following steps:

1. All clusters are structured as a time series based on the time they were created.
2. All clusters are given zero timestamps.

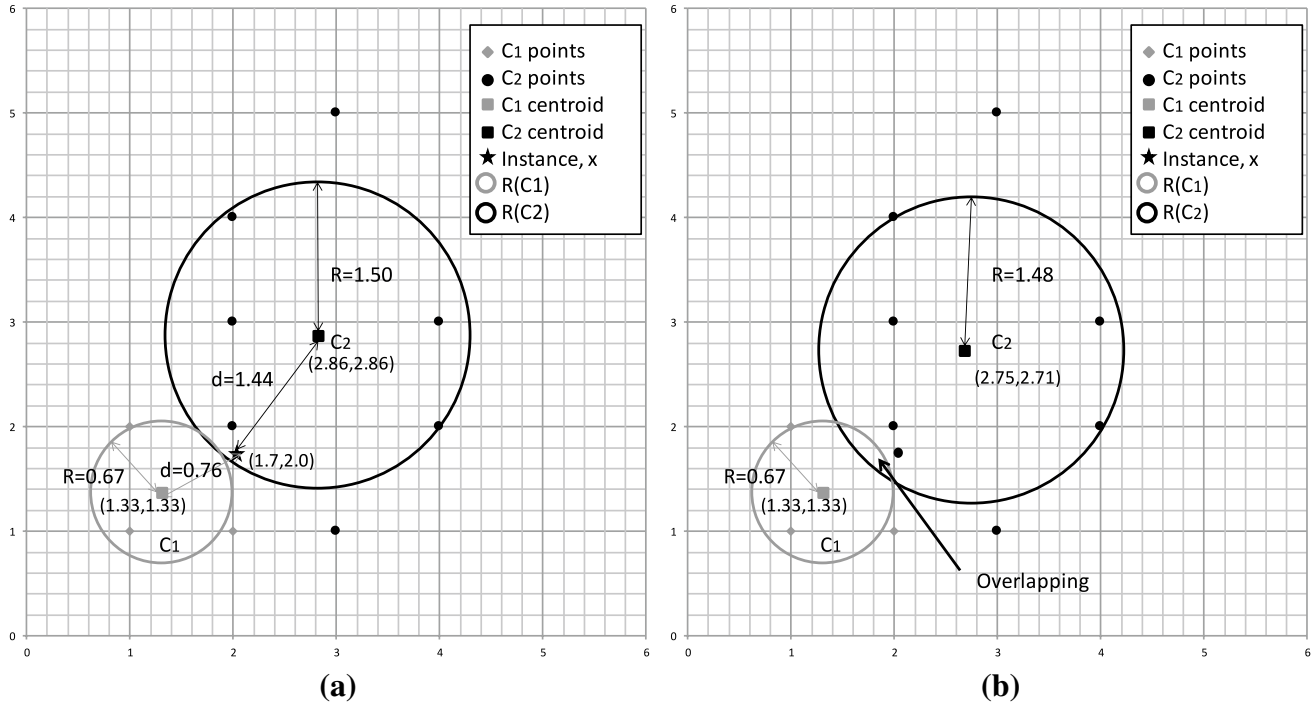

Fig. 2 Example of boundary condition **a** before model merging with C_2 **b** after merging with C_2

3. When a cluster is nominated as the nearest cluster in the classification process, the timestamp is incremented by one.
4. When a chunk is received, the timestamps of each cluster is checked from the beginning of the series. Clusters with timestamps zero will be deleted until the number of cluster, k is reduced to a user-predefined number, $\frac{k_d}{B}$. Then, the timestamps of remaining clusters will be decremented by one. This step repeats until $k \leq \frac{k_d}{B}$.

After the deletion of unused clusters, the remaining clusters will be re-partitioned into B micro-models based on their centroid locations in the Euclidean space.

4 Results and discussion

This section describes the simulation setup and results of our proposed work. The experiment is conducted to explore the ability of the online traffic classifier to learn accurately using online features even in the presence of concept drift.

4.1 Datasets

Two real traffic datasets; UNIBS (Gringoli et al. 2009) and Cambridge (Moore et al. 2005) are chosen for the experiment. The UNIBS dataset was captured in University of Brescia. It was collected in three consecutive days and the traces are in pcap format and come with a groundtruth. The traces were processed to extract online features with only the first five packets of each observed flow as in Monemi et al. (2013). By using the provided groundtruth labels, we labeled the flow into six classes (Web, Mail, P2P, SKYPE, MSN and Others). The Cambridge dataset was captured from University of Cambridge network. It contains 248 flow features and 12 classes. A total of 11 features are selected from the dataset as discussed in Sect. 4.2. The data of the minimal class, games and interactive are not used as it is not sufficient for training and testing (less than 10 flow instances). The details of used datasets are summarized in Table 3, and the selection of features are described in Sect. 4.2.

4.2 Extraction of online features

In order to perform online classification and learning, flow features that can be extracted from live traffic are selected. Although Moore et al. (2005) suggested 248 flow features that can be potentially used in machine learning traffic classification, most of them could not be extracted online. Features such as minimum bytes in packet, maximum bytes in packet, and median bytes in packet only can be extracted after receiving complete flows. Considering this type of features in classification will delay traffic classification decision until the end of each flow. Hence, this approach cannot be use to classify live network traffic. Based on

Table 3 Dataset used for experimental works

	UNIBS (Gringoli et al. 2009)	Cambridge (Moore et al. 2005)	
	Extracted	Original	Selected
# Flow features	11	248	11
# classes	6	12	10
# Flow instances	77303	397,152	397,030

works by Monemi et al. (2013), we can categorize the types of online features as follow:

1. *Header features:* Information that can be collected from packet header can be consider as online features, since it is available as early as in the first packet such as source port and destination port.
2. *First n packets features:* The statistical information that can be computed using only first few packets, such as first n IP packets size, and first n Ethernet frames size. It can be computed before the flow ended.

The features that are selected for the classification process are shown in Table 4. As most of the features in Cambridge dataset are not online, we assume those first quartile features to be the first n packets statistical features for comparison with the UNIBS dataset. Online features extraction will not be the bottleneck of online data stream classification as Monemi et al. (2013) has proposed a NetFPGA based traffic classifier that can extract traffic features at line speed. Note that we do not consider packet inter-arrival time as a part of online features although this feature is listed as real-time or on-line features in Monemi et al. (2013). This is due to the fact that inter-arrival time need pre-processing such as normalization in order to become significant. In addition, references (Erman et al. 2007; Zhen and Qiong 2012) have reported that the use of time-related features does not result in improved accuracy. We also perform experiments to show the impact of using packet inter-arrival time on the accuracy of the proposed system. We will discuss this in Sect. 4.4.

4.3 Experimental setup

The model parameters used in our experiment are as stated below, unless specified otherwise:

1. Percentage of labeling, $P = 10\ \%$
2. Chunk size, $N_{\text{chunk}} = 1000$
3. Number of micro-model, $B = 10$
4. Desired number of cluster, $k_d = 100$
5. Boundary threshold $= 2$.

In our experiment, the first chunk of data (first 1000 instances) are treated as pre-collected flow instances and they are used for model initialization. The rest of the data are randomly labeled for different percentage P. For experiment without incremental learning, the selection of learning instances (Sect. 3.3) and the whole semi-supervised incremental learning (Sect. 3.4) will be not be executed. The accuracy of the proposed model is verified using the interleaved test-then-train or prequential method where each flow instance was first tested before being trained incrementally (Bifet et al. 2011). Each experiment was repeated 100 times, and the average performance indicators are reported in this paper.

The performance indicators used in this paper are immediate accuracy (Acc_i), cumulative accuracy (Acc_c), average accuracy (Acc_{avg}), running time (T_{run}), classification time (T_c), injecting time (T_i), reconstruction time (T_{con}) and memory requirement. Three types of accuracy measures refer to the accuracy of classification model at different

Table 4 List of online features selected for online classification

ID	Name	Description	Description as in (Moore et al. 2005)
01	Source Port	Source port number	Server Port
02	Destination Port	Destination port number	Client Port
03	TL IP	Total bytes in IP packet	q1_data_ip
04	UL IP	Total bytes in IP packet (uplink)	q1_data_ip_a b
05	DL IP	Total bytes in IP packet (downlink)	q1_data_ip_b a
06	TL Eth	Total bytes in Ethernet packet	q1_data_wire
07	UL Eth	Total bytes in Ethernet packet (uplink)	q1_data_wire_a b
08	DL Eth	Total bytes in Ethernet packet (downlink)	q1_data_wire_b a
09	TL Ctr	Total control bytes in packet	q1_data_control
10	UL Ctr	Total control bytes in packet (uplink)	q1_data_control_a b
11	DL Ctr	Total control bytes in packet (downlink)	q1_data_control_b a

We assume features with prefix q1 (first quartile) are similar to n packet statistical features as the Cambridge dataset does not contain actual packet traces

Table 5 Packet inter-arrival time related features

Description	UNIBS	Cambridge
Total packet inter-arrival time	Inter-arrival time for the first five packets	*q1_IAT*
Souce to destination packet inter-arrival time	Inter-arrival time for first five uplink packets	*q1_IAT_a b*
Destination to source packet inter-arrival time	Inter-arrival time for first five downlink packets	*q1_IAT_b a*

points. Let the total correct classification in a chunk of N_{chunk} instances is η_{chunk}, Acc_i and Acc_c at f-th chunk is:

$$Acc_i(f) = \frac{\eta_{\text{chunk}}}{N_{\text{chunk}}} \times 100\ \% \tag{7}$$

$$Acc_c(f) = \frac{\sum_{i=1}^{f} Acc_i}{f} \tag{8}$$

while Acc_{avg} is Acc_c for the last chunk. T_{run} is defined as the total running time for one chunk. It includes the time to classify one chunk, time for injecting labeled instances and time for model reconstruction. In our experiment, the T_{run} does not include data labeling time as in (Masud et al. 2012), since data labeling are usually done offline and is beyond the scope of discussion in this paper. T_c is measured based on the average time needed to classify one chunk and not including feature extraction time. T_i is the average time to inject a labeled instance while T_{con} is measured on the time to complete micro-model reconstruction step, which include the cluster reduction and repartitioning time.

4.4 Impact of packet inter-arrival time

As discussed previously in Sect. 4.2, packet inter-arrival time as a time related feature does not give additional benefit, as reported in Erman et al. (2007); Zhen and Qiong 2012). We evaluated the performance of different features set with and without packet inter-arrival time in our proposed algorithm for both UNIBS and Cambridge datasets. Features used are as listed in Table 5.

We first tested the algorithm to include packet inter-arrival time without modifying these features. As these features are one million times smaller value than the other features, it does not make much difference in the accuracy. The different in Acc_i of using features without packet inter-arrival time compared to with packet inter-arrival time is plotted in Fig. 3, where

$$\Delta Acc_i = Acc_{i-WOIAT} - Acc_{i-WIAT} \tag{9}$$

In the second set of experiments, we represent the packet inter-arrival time in µs instead in second (s). Figure 4 shows that, with modified packet inter-arrival time, the accuracy difference for UNIBS dataset is at 5 % average. This indicates that with the use of inter-arrival time in µs will reduce the accuracy. With this, we decided not to include packet inter-arrival time in our further works.

4.5 Performance

This subsection discusses the overall performance of the proposed algorithm. The accuracy in time-series is shown in Fig. 5. As we only use the first chunk in the dataset for model initialization, the classes which were not seen in the first chunk are treated as new concepts. A drift detection experiment was conducted on our dataset by using Drift Detection Method (Gama et al. 2004) provided in MOA tools (Waikato 2015). A series of drifts were detected and we plotted them in Fig. 6. We found out that Cambridge dataset has more detected drifts than the UNIBS dataset. In order to visualize the drifts clearly, we highlighted in Fig. 6, the Cambridge dataset showing chunks 150 to 250, as this range shows both areas with drift and without drifts. In the figure, we observed that when concept drifts occur (Drift detected = 1), our proposed algorithm with incremental learning can learn from the new knowledge and is able to maintain the accuracy compared to the model without incremental learning.

Table 6 shows the overall performance of our proposed algorithm. In this paper, we assume that the classifier is the bottleneck of the network traffic classification system, as reported in several works such as (Monemi et al. 2013), flow features extraction in FPGA can function in very high speed. We compute the performance of the classification in terms of milisecond per flow instances in software as is the lower bound of the system performance. The classification time of our proposed algorithm is approximate 4 ms to classify 1000 flow instances. Besides, we found that the time for injecting a labeled flow instance was similar to the time for classifying a new flow instance. Thus, injecting labeled flow instance will not cause additional delay and affect the overall online classification process. The time for model reconstruction ranges from 3 to 5 % of the total running time. This delay can be further reduced by having the model reconstruction done outside the model and replace it once reconstruction is completed. Our proposed system does not require large memory as only summary of cluster's information are kept. It only requires less than 1 kB RAM-hour to complete the processes.

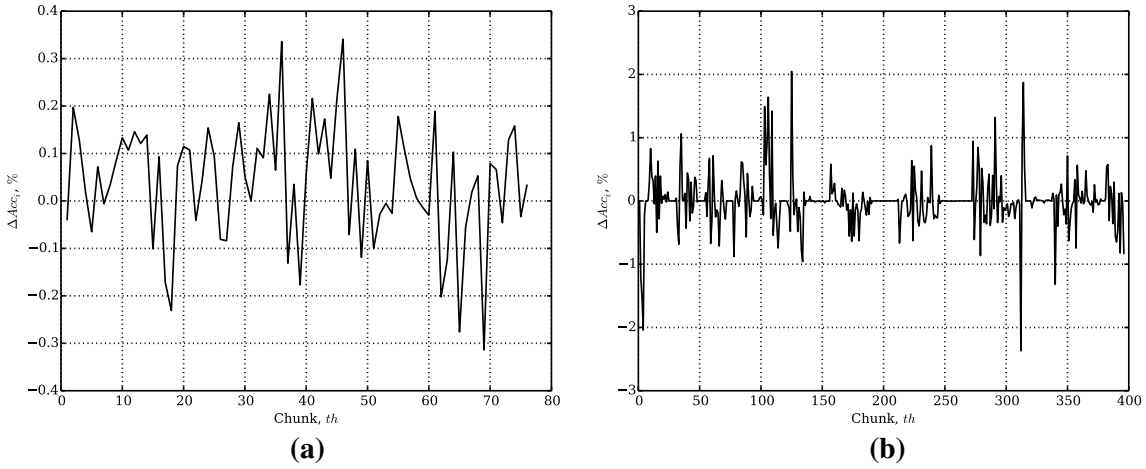

Fig. 3 Immediate accuracy difference ΔAcc_i with original packet inter-arrival time using **a** UNIBS, **b** Cambridge datasets

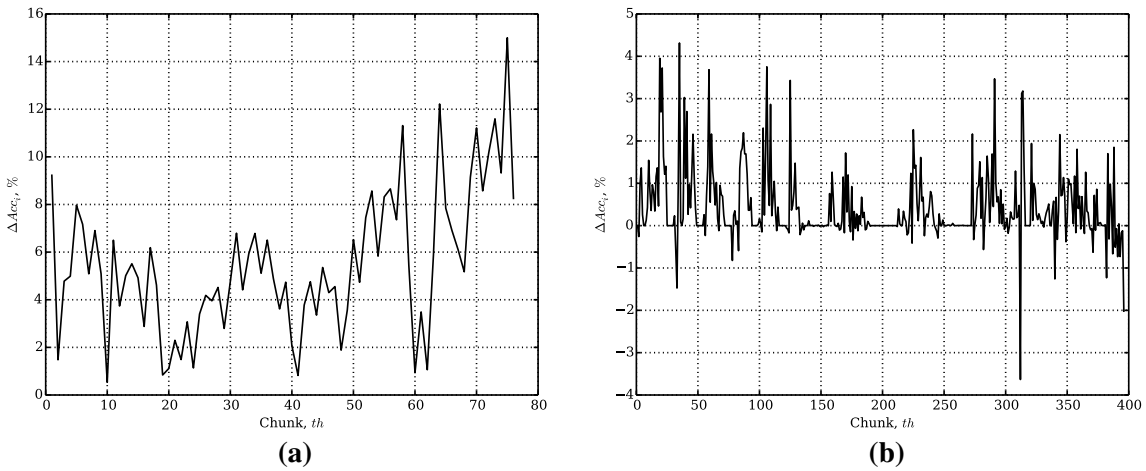

Fig. 4 Immediate accuracy difference ΔAcc_i with modified packet inter-arrival time using **a** UNIBS, **b** Cambridge datasets

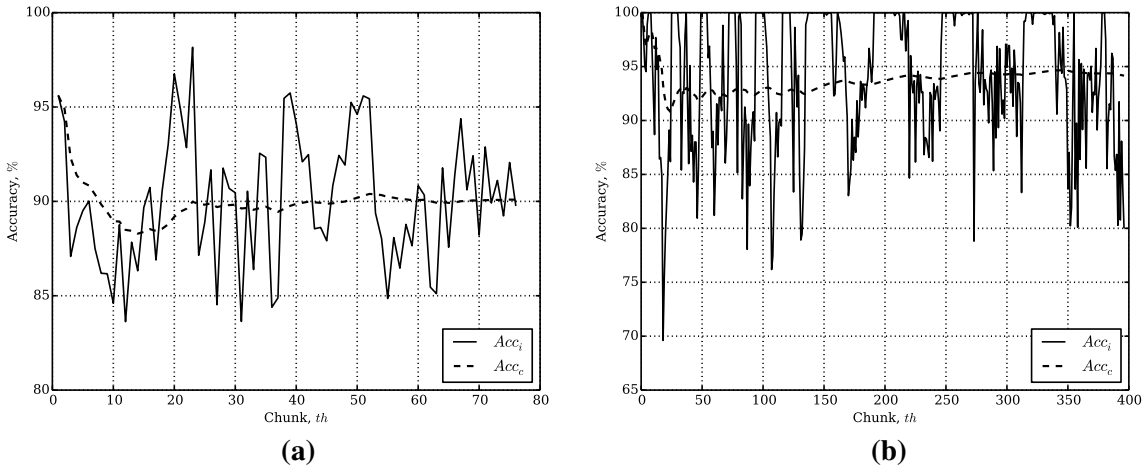

Fig. 5 Accuracy of the classification model on **a** UNIBS, **b** Cambridge datasets

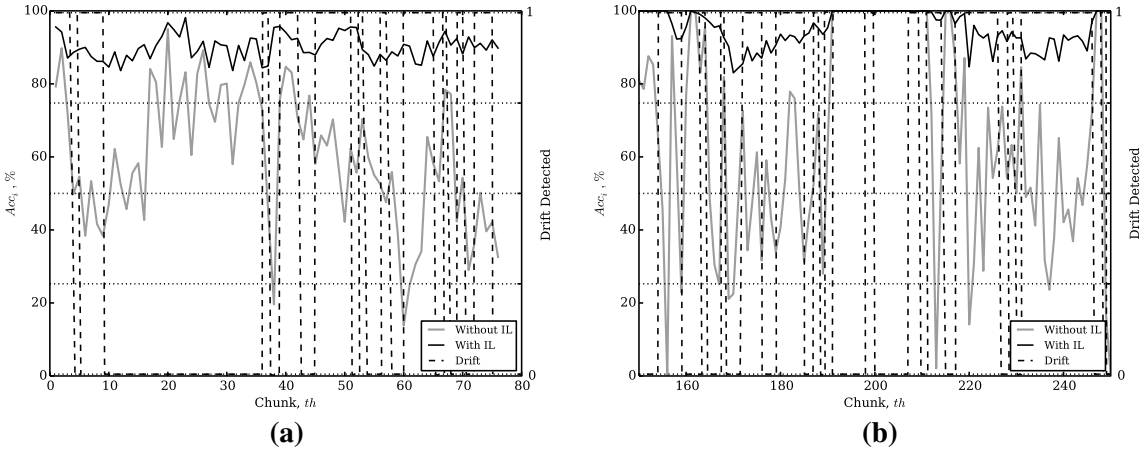

Fig. 6 Immediate accuracy comparison on the effect of incremental learning of the classification model during Concept Drift on **a** UNIBS, **b** Cambridge datasets. For Cambridge dataset, we show only 150th–250th chunks to illustrate the effect of incremental learning. *Vertical dashed lines* indicate the time when concept drifts occur according to Drift Detection Method (Gama et al. 2004)

Table 6 Overall performance

Dataset	UNIBS	Cambridge
Average accuracy, Acc_{avg} (%)	91.28	94.38
Running time, T_{run} (ms/1000 flow instances)	3.63	5.07
Classification time, T_c (ms/1000 flow instances)	3.06	4.39
Injecting time, T_i (µs/labeled instances)	3.49	4.62
Reconstruction time, T_{con} (ms/chunk)	0.19	0.17
RAM-hour (kB)	0.02	0.14

4.6 Impact of different parameters setting on classification performance

In this subsection, we analyze the effect of changing algorithm parameters on system performance. The experiments are done by changing one parameter and fixing other parameters. Figure 7 shows how labeling percentage, P is affecting the average accuracy and running time. Increases in labeled flows provide more class information to the classification model for better accuracy. However, we also noted that with $P = 100$ %, the maximum accuracy of the classification model are not as high as those in fully supervised batch learning method. Our proposed method uses incremental method for learning, where it does not have full visibility on dataset as in batch learning. The latter cannot be used for classifying traffic with concept drifts as batch learning assume all data are pre-collected and this assumption is not applicable for data streams classification such as for network traffic. The percentage of labeling does not affect classification time, but it increases the running time. As more labeled flows detected, longer time need for injecting labeled flow instance.

The number of micro-model, B determines the number of partitions that the classification model will be split into. Larger B results in more partitions in the classification model. Hence, the clusters involved in estimating the nearest neighbor during classification will be reduced. However, the increase in B will also affect accuracy. The more the classification model is split into, the less information it can view from a particular micro-model perspective. Hence small trade-off in accuracy is expected in order to achieve high efficiency. Figure 8 shows the impact of changing number of micro-model towards accuracy and classification time.

The number of desired clusters k_d is the parameter used in micro-model reconstruction process. It is used to maintain the number of clusters in the classification model. However, any change of this parameter does not cause a significant impact on the overall performance. It will only cause slight increase in the reconstruction time because the model will need to go through a lot of clusters in the reconstruction process to repartition the micro-model when k_d is large. However, this increase does not have significant impact on overall running time as the reconstruction process constitute to only to be 3–5 % of the overall running time. Figure 9 shows the impact of different number of desired clusters on accuracy and reconstruction time.

Similar to window size, chunk size in our proposed method will determine how fast the model can undergo reconstruction. This parameter is dependent on the types of dataset. Figure 10 shows the behavior of performance with the changes of chunk size. We can observe different trend in performance for UNIBS and Cambridge datasets. The UNIBS dataset has less drifts and the use of bigger chunk size is more appropriate. The Cambridge dataset contains

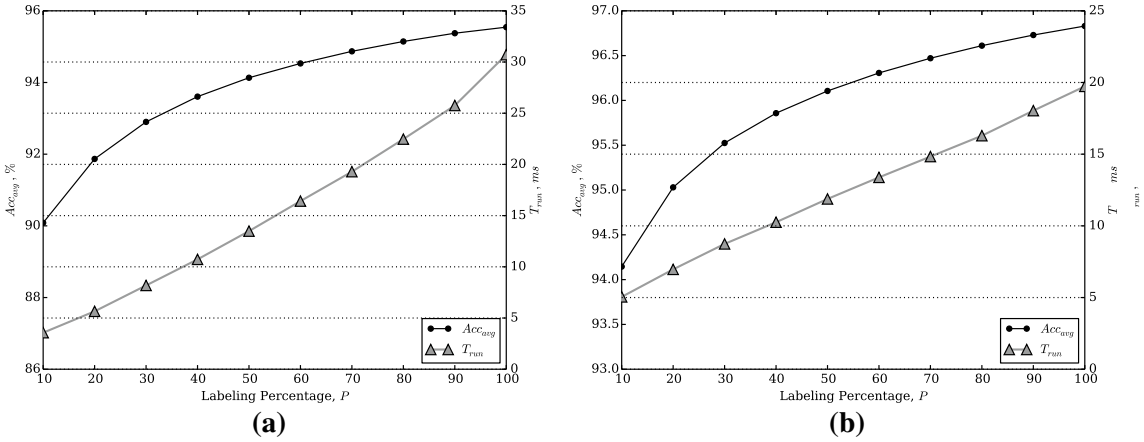

Fig. 7 The impact of percentage of flow instance labeling, P on accuracy (Acc_{avg}) and running time (T_{run}) **a** UNIBS, **b** Cambridge datasets

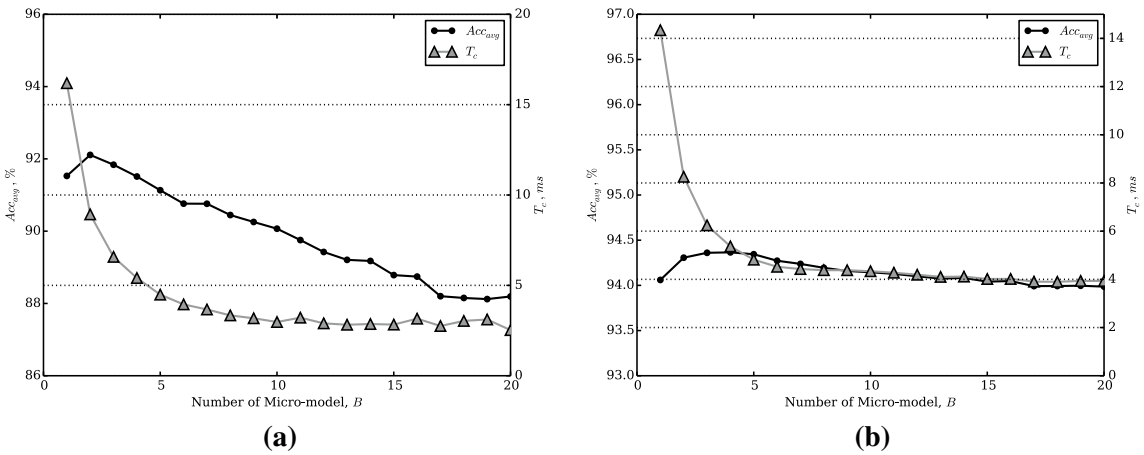

Fig. 8 The impact of number of micro-model, B on accuracy (Acc_{avg}) and classification time (T_c) **a** UNIBS, **b** Cambridge datasets

more drifts as discussed in Sect. 4.5. Thus, small chunk size allows faster unlearning and thus, results in better performance.

4.7 Performance comparison with other methods

In this section, we look at the comparison between our proposed method and other methods. We selected several algorithms implemented in MOA (Waikato 2015) based on the suggestion by Bifet et al. (2009, 2013); de Souza et al. 2014). The selected algorithms are as follows:

1. ADWIN Bagging using Hoeffding Adaptive Trees (Bag ADWIN 10 HAT) by Bifet et al. (2009)
2. Bagging ASHT using the DDM drift detection method (DDM Bag 10 ASHT W) by Bifet et al. (2009)
3. Levaranging Bagging with K-Nearest Neighbour and Probabilistic Adaptive Windowing (LB KNN W) by Bifet et al. (2013)
4. OzaBoost Dynamic with two classifiers (VFDT and Naive Bayes Multinomial-NBM) (OzaDyn) by de Souza et al. (2014).

In order to provide fair comparisons, these algorithms were executed in the same computer system using identical evaluation method (Prequential Method) and identical datasets (UNIBS and Cambridge). All mentioned algorithms were executed in MOA using the default settings. Since all the above mentioned algorithms are not semi-supervised method, the comparison of these methods with our proposed method were done by using fully labeled method, $P = 100$. The comparison results are shown in Table 7. From the results, although our proposed method does not provide the highest accuracy in the Cambridge dataset, but it is have the shortest running time. Figure 11 shows the accuracy comparison over time. It is clear that for UNIBS dataset, our proposed method can maintain accuracy better than other methods.

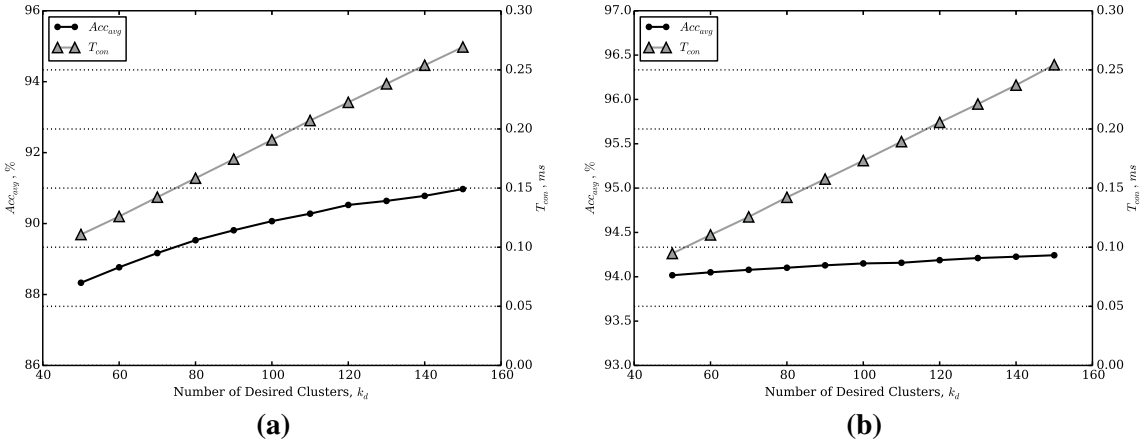

Fig. 9 The impact of different number of desired cluster, k_d on accuracy (Acc_{avg}) and reconstruction time (T_{con}) **a** UNIBS, **b** Cambridge datasets

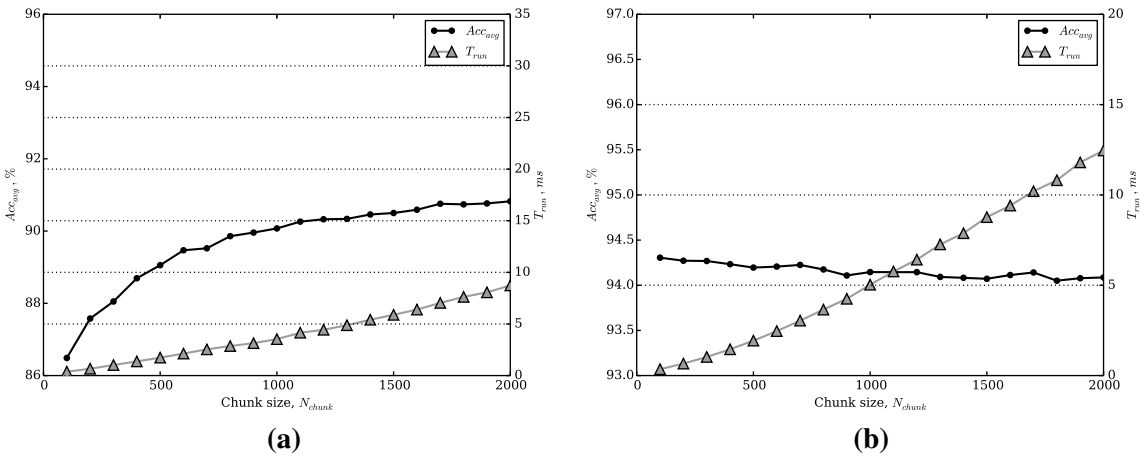

Fig. 10 The impact of different chunk sizes, N_{chunk} on accuracy (Acc_{avg}) and running time (T_{run}) **a** UNIBS, **b** Cambridge datasets

4.8 Performance comparison with other semi-supervised learning method

In order to compare our proposed method with the method reported by Liu et al. (2014), Masud et al. (2012), Bertini Jr et al. (2012), we run our proposed method using the real-trace datasets based on each of the paper. KDD'99 dataset was used in ReaSC by Masud et al. (2012) and ECMBDF by Liu et al. (2014). In this dataset, as our classification model only process numerical attributes, we select only numerical attributes and categorical attributes are ignored, as in Masud et al. (2012), resulting a dataset that consist of 34 attributes and 23 classes. KAOGINCSSL by Bertini Jr et al. (2012) uses real dataset Elec2 and Spam which consist of five attributes and seven attributes, respectively. Both datasets have two classes. By following the parameter setting and distribution of chunk as stated in these papers, we compare the average accuracy for 10 % labeled instances. The accuracy comparison on these datasets are shown in Table 8. The

Table 7 Performance comparison

	Proposed method	*a*	*b*	*c*	*d*
(a) UNIBS datasets					
Average accuracy (%)	**95.58**	94.50	94.73	94.84	87.74
Overall running time (s)	**2.58**	7.47	6.05	312.04	9.72
RAM-hour (kB)	**0.22**	0.72	1.14	201.27	0.38
(b) Cambridge datasets					
Average accuracy (%)	96.86	**98.94**	98.30	96.85	97.48
Overall running time (s)	**10.76**	29.51	29.58	2424.70	66.22
RAM-hour (kB)	**0.90**	2.18	2.44	1534.83	3.87

Bold values are the best among the comparison

[a] Bag ADWIN 10 HAT (Bifet et al. 2009)

[b] DDM Bag 10 ASHT W (Bifet et al. 2009)

[c] LB KNN W (Bifet et al. 2013)

[d] OzaDyn (de Souza et al. 2014)

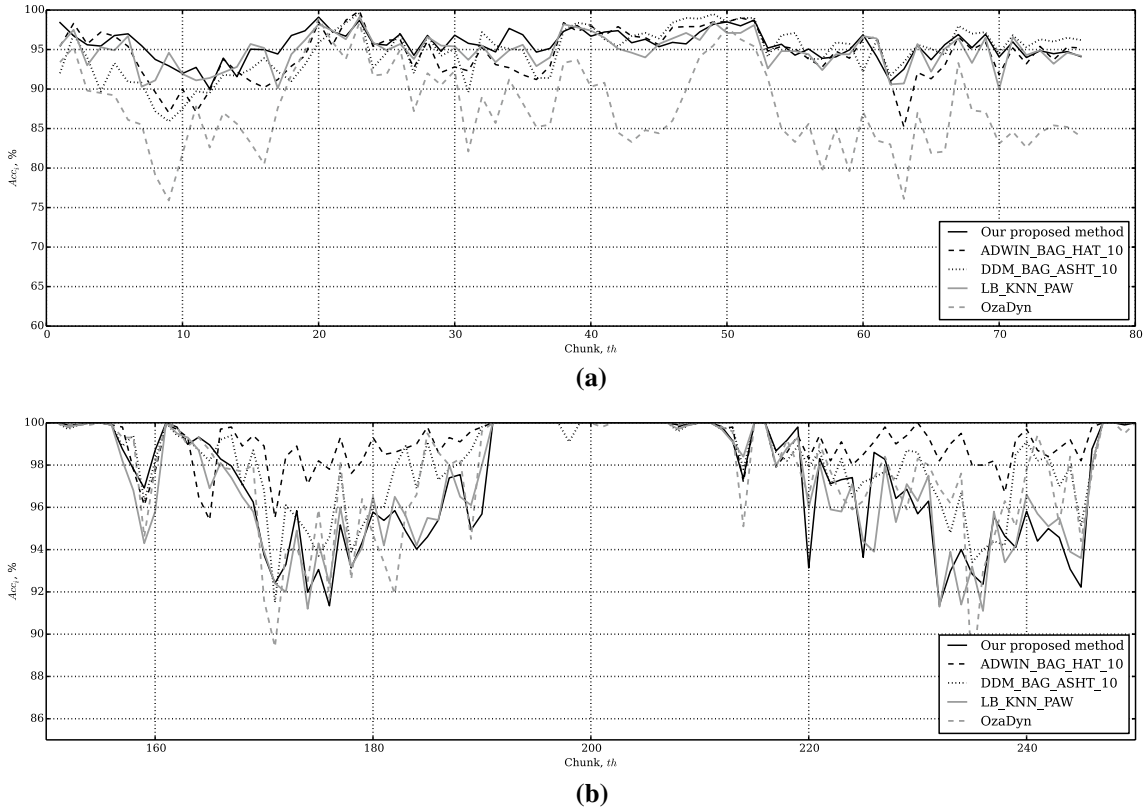

Fig. 11 Comparison between proposed method and other methods on **a** UNIBS and **b** Cambridge datasets. For Cambridge dataset, we only show from chunk 150 to chunk 250 to illustrate the difference of accuracy between our proposed method and other prior works

Table 8 Accuracy comparison with other semi-supervised learning method

Dataset	Proposed method	ReaSC	ECMBDF	KAOGINCSSL
KDD'99	**99.51**	96.2	90.89	–
Elec2	**78.93**	–	–	56.54
Spam	**80.72**	–	–	75.79

Bold values are the best among the comparison

Table 9 Performance comparison with ReaSC

Method	Proposed method	ReaSC
Running time (s/1000 instances)	**0.008**	0.83
Classification time (s/1000 instances)	**0.007**	0.36

Bold values are the best among the comparison

performance comparison with ReaSC is shown in Table 9. We are only able to perform the performance comparison (Table 9) with ReaSC as the performance result was not reported in other works. The results show that our proposed method achieves higher accuracy and better running time.

4.9 Discussion

The results show that the proposed method with incremental learning outperforms non-incremental learning in terms of accuracy and performance for classification of data stream. Non-incremental learning uses the early chunks of stream to built k_d clusters. They are static and will not change throughout the classification process. As data streams are time-series, the data distribution evolves over time which may be caused by environments and external factors. They may introduce new concept or even new classes. We also show this in Sect. 4.5, drifts are detected in both datasets. When drifts occur, the classification model that was trained with early stream chunks does not have the knowledge of new concept hence it fails to maintain the accuracy. On the other hand, the incremental learning can provide continuous update of current data distribution to the classification model. The clusters continuously grows with new knowledge and periodically the classification model can reduce outdated clusters that are no longer carry the current information. This self-adapting characteristic reduce the false classification that may be caused by outdated knowledge and hence, able to maintain the reliability and correctness of the classification model.

5 Conclusion

This paper proposed and analyzed an online incremental learning algorithm for network traffic classification. By applying semi-supervised techniques on incremental *k*-means, our proposed method are able to adapt to the concept drift with only 10 % of labeled flow instances. Our results show that: (1) high accuracy can be achieved even with the use of only eleven features that are able to be extracted online; (2) incremental learning provide higher accuracy the method without incremental learning; (3) lower running time compared to other data stream mining methods; and, (4) lower RAM usage compared to other data stream mining methods. In future, we will implement this work in reprogrammable hardware so that it can perform inline classification on live traffic.

References

Abdulsalam H (2008) Streaming Random Forest. PhD thesis, School of Computing, Queen's University, Kingston, Ontario, Canada

Aggarwal CC, Jiawei H, Jianyong W, Philip SY (2004) On demand classification of data streams. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'04, New York, NY, USA. ACM, pp 503–508

Aggarwal Charu C, Han Jiawei, Wang Jianyong, Yu Philip S (2006) A framework for on-demand classification of evolving data streams. IEEE Trans Knowl Data Eng 18(5):577–588

Angelov Plamen P, Zhou Xiaowei (2008) Evolving fuzzy-rule-based classifiers from data streams. IEEE Trans Fuzzy Syst 16(6):1462–1475

Baena-García M, José del Campo-Ávila J, Raúl F, Albert B, Gavaldà R, Morales-Bueno R (2006) Early drift detection method. 6:77–86

Bertini Jr, João R, de Andrade Alneu, Lopes AA, Liang Z (2012) Partially labeled data stream classification with the semi-supervised K-associated graph. J Braz Comp Soc 18(4):299–310

Bifet A, Holmes G, Kirkby R, Pfahringer B (2011) Data stream mining: a practical approach. Technical report, University of Waikato

Bifet A, Holmes G, Pfahringer B, Gavalda R (2009) Improving adaptive bagging methods for evolving data streams. In: Advances in Machine Learning. Springer, pp 23–37

Bifet A, Pfahringer B, Read J, Holmes G (2013) Efficient data stream classification via probabilistic adaptive windows. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp 801–806

Dainotti Alberto, Pescape Antonio, Claffy Kimberly C (2012) Issues and future directions in traffic classification. IEEE Netw 26(1):35–40

de Souza EN, Matwin S, Fernandes S (2014) Traffic classification with on-line ensemble method. In: Global Information Infrastructure and Networking Symposium (GIIS), IEEE, pp 1–4

Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'00, New York, NY, USA, ACM, pp 71–80

Erman Jeffrey, Mahanti Anirban, Arlitt Martin, Cohen Ira, Williamson Carey (2007) Offline/realtime traffic classification using semi-supervised learning. Perform Eval 64(9):1194–1213

Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: Advances in Artificial Intelligence-SBIA 2004, Springer, pp 286–295

Gringoli F, Salgarelli L, Cascarano N, Risso F, Claffy KC (2009) GT: picking up the truth from the ground for internet traffic. ACM SIGCOMM Comp Commun Rev 39:13–18

Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'01, New York, NY, USA, ACM, pp 97–106

Li S (2012) Towards ultra high-speed online network traffic classifcation enhanced with machine learning algorithms and openflow accelerators. PhD thesis, University of Massachusetts Lowell

Li W, Moore AW (2007) (2007) A machine learning approach for efficient traffic classification. In: 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, IEEE, MASCOTS'07., pp 310–317

Liu Jing, Guo-sheng Xu, Zheng Shi-hui, Xiao Da, Li-ze Gu (2014) Data streams classification with ensemble model based on decision-feedback. J China Univ Posts Telecommun 21(1):79–85

Loo HR, Andromeda Trias, Marsono MN (2014) Online data stream learning and classification with limited labels. Proc Elect Eng Comp Sci Inform 1(1):161–164

Loo HR, Marsono MN (2015) Online data stream classification with incremental semi-supervised learning. In: 2nd IKDD Conference on Data Science, CODS'15, ACM, pp 132–133

Lughofer E, Angelov P (2009) Detecting and reacting on drifts and shifts in on-line data streams with evolving fuzzy systems. In: Proceedings of the Joint 2009 International Fuzzy Systems Association World Congress and 2009 European Society of Fuzzy Logic and Technology Conference, Lisbon, Portugal, July 20–24, IFSA, Lisbon, pp 931–937

Lughofer Edwin, Angelov Plamen (2011) Handling drifts and shifts in on-line data streams with evolving fuzzy systems. Appl Soft Comp 11(2):2057–2068

Masud Mohammad M, Woolam Clay, Gao Jing, Khan Latifur, Han Jiawei, Hamlen Kevin W, Oza Nikunj C (2012) Facing the reality of data stream classification: coping with scarcity of labeled data. Knowl Inform Syst 33(1):213–244

Mingliang G, Xiaohong H, Xu T, Ma Y, Zhenhua W (2009) Data stream mIning based real-time high speed traffic classification. In: Proceedings of 2nd IEEE International Conference on Broadband Network and Multimedia Technology, 2009. IC-BNMT'09., pp 700–705

Minku L, Yao Xin (2012) DDD: a new ensemble approach for dealing with concept drift. IEEE Trans Knowl Data Eng 24(4):619–633

Monemi Alireza, Zarei Roozbeh, Marsono Muhammad Nadzir (2013) Online NetFPGA decision tree statistical traffic classifier. Comp Commun 36(12):1329–1340

Moore A, Zuev D, Crogan M (2005) Discriminators for use in flow-based classification. Technical report, Department of Computer Science, Queen Mary, University of London

Qian Feng, Guang-min Hu, Yao Xing-miao (2008) Semi-supervised internet network traffic classification using a Gaussian mixture model. AEU Int J Elect Commun 62(7):557–564

Raahemi B, Mumtaz A (2008) A two-stage window-based architecture for classification of peer-to-peer traffic using fast decision tree. In: Proceedings of the 4th International conference on data mining DMIN2008. Las Vegas, Nevada, USA, pp 144–149

Raahemi B, Zhong W, Liu J (2008) Peer-to-peer traffic identification by mining IP layer data streams using concept-adapting very fast decision tree. In: 20th IEEE Internationtal Conference on Tools with Artificial Intelligence, vol 1, pp 525–532

Shrivastav A, Tiwari J (2010) Network traffic classification using semi-supervised approach. In: IEEE 2010 Second International

Conference on Machine Learning and Computing (ICMLC), pp 345–349

Street WN, Kim YS (2001) A streaming ensemble algorithm (SEA) for large-scale classification. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'01, New York, NY, USA. ACM, pp 377–382

Tian X, Sun Q, Huang X, Ma Y (2008) Dynamic online traffic classification using data stream mining. In: International Conference on MultiMedia and Information Technology, MMIT'08, IEEE, pp 104–107

Waikato (2015) MOA massive online analysis. http://moa.cs.waikato.ac.nz/

Wang H, Fan W, Yu PS, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '03, New York, NY, USA. ACM, pp 226–235

Zhang T, Raghu R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. In: ACM SIGMOD Record, ACM, vol 25, pp 103–114

Zhen Liu, Qiong Liu (2012) A new feature selection method for internet traffic classification using ml. Phys Procedia 33:1338–1345