Soft Comput (2017) 21:2035–2046
DOI 10.1007/s00500-015-1902-3

METHODOLOGIES AND APPLICATION

Flexible neural trees based early stage identification for IP traffic

Zhenxiang Chen[1,2] · **Lizhi Peng**[1,2] · **Chongzhi Gao**[3] · **Bo Yang**[1,2] · **Yuehui Chen**[1,2] · **Jin Li**[3]

Published online: 26 October 2015

Abstract Identifying network traffics at their early stages accurately is very important for network management and security. Recent years, more and more studies have devoted to find effective machine learning models to identify traffics with few packets at the early stage. In this paper, we try to build an effective early stage traffic identification model by applying flexible neural trees (FNT). Three network traffic data sets including two open data sets are used for the study. We first extract both packet-level features and statistical features from the first six continuous packets and six noncontinuous packets of each flow. Packet sizes are applied as packet-level features. And for statistical features, average, standard deviation, maximum and minimum are selected. Eight classical classifiers are employed as the comparing methods in the identification experiments. Accuracy, true positive rate (TPR) and false positive rate (FPR) are applied to evaluate the performances of the compared methods. FNT outperforms the other methods for most cases in the identification experiments, and it behaves very well for both TPR and FPR. Furthermore, it can show the selected features in the optimal tree result. Experiment result shows that FNT is effective for early stage traffic identification.

Keywords Early stage traffic identification · Flexible neural trees · Machine learning

Communicated by V. Loia.

✉ Lizhi Peng
penglizhi.jn@gmail.com

Zhenxiang Chen
czx@ujn.edu.cn

Chongzhi Gao
gaochongzhi@gzhu.edu.cn

Bo Yang
yangbo@ujn.edu.cn

Yuehui Chen
yhchen@ujn.edu.cn

Jin Li
lijin@gzhu.edu.cn

[1] School of Information Science Engineering, University of Jinan, Jinan 250022, People's Republic of China

[2] Shandong Provincial Key Lab of Network based Intelligent Computing, Jinan 250022, People's Republic of China

[3] Department of Computer Science, Guangzhou University, Guangzhou 510006, Guangdong Province, People's Republic of China

1 Introduction

Accurate network traffic identification is an important way to ensure network security and to provide good quality of service. In general, port-based technique and deep packet inspection are two traditional traffic identification methods. However, such traditional methods become invalid when confronting dynamic port numbers and encrypted traffic in modern Internet environment; therefore, machine-learning-based traffic identification has attracted much researching interests in the last decade. Most traditional machine-learning-based traffic identification techniques extract features on a whole traffic instance (Este et al. 2009b; Li and Moore 2007; Moore and Zuev 2005; Dainottia et al. 2008). The most extracting method is presented by Moore et al. in (2005). They extract 248 statistical features based on whole traffics, such as maximum, minimum and average values of packet size, RTT. And classifiers using these statistical features can get quite high performances in traffic identification.

However, in real circumstances, it makes no sense to recognize Internet traffics when they have ended. In recent

years, the early stage traffic identification has caught enough interests at the research community. In real cases, we must identify Internet traffics accurately in their early stage so that we can apply subsequent management and security policies. Therefore, some researchers have turned to find effective models which are able to identify Internet traffics at their early stage. And this makes the early stage identification become a hot topic in traffic identification researches (Dainotti et al. 2012; Palmieri and Fiore 2009; Esposito et al. 2013; Zhang et al. 2014). Qu et al. (2012) have studied the problem of the accuracy of early stage traffic identification, and found that it is possible to identify traffic accurately at its early stage.

In this paper, we set out to create an effective early stage traffic identification model by applying flexible neural trees. Three network traffic data sets including two open data sets are used for the study. We first extract both packet-level features and statistical features from the first six continuous packets and six noncontinuous packets of each flow. Packet sizes are applied as packet-level features. And for statistical features, average, standard deviation, maximum and minimum are selected. Eight classical classifiers are employed as the comparing methods in the identification experiments. FNT shows its high performances for most cases. The experimental results also confirm that a continuous packet sequence is more effective than a noncontinuous one. However, the early noncontinuous six packets can still help to identify the IP traffic correctly in some extent, which can greatly meet the identification requirement in real world.

The remainder of the paper is organized as follows. In Sect. 3, some related works about the current research achievement of the early stage traffic classification and the FNT method are reviewed. The detailed description of the proposed approach of the FNT-based early stage classification is presented in Sect. 4. In Sect. 5, the experiment approach is addressed. The experiment result and analysis are addressed in Sect. 6. In Sect. 7, we design and build a real-world implementation early stage identification structure. We devote the final section to some key concluding remarks and future works.

2 Related work

It is relatively hard to recognize a traffic by only using several early stage packets. Thus, the key problem of the early stage traffic identification is to find out effective features in the early stage of traffics. Bernaille et al. presented a famous early stage traffic identification technique in (2006). They use the size of the first few data packets of each TCP flow as the features, and by applying K-means clustering technique, they get high identification rates for 10 types of application traffics. Este et al. proved in (2009a) that early stage packets of an Internet flow carried enough information for traffic classification. They analyzed round trip time (RTT), packet size, inter-arrival time (IAT) and packet direction of the early stage packets and found that packet size was the most effective feature for early stage classifications. Huang et al. studied the early stage application characteristics and used them for classification effectively in (2008). Recently, they extracted the early stage traffic features by analyzing the negotiation behaviors of different applications. They use packet size (PS) and inter packet time (IPT) of the first 10 packets for some classifiers, while for other classifiers, they use average and standard deviation values of PS and IPT of the early packets. They applied these features for machine-learning-based classifiers with high performances (Huang et al. 2013).

Hullár et al. proposed an automatic machine-learning-based method consuming limited computational and memory resources for P2P traffic identification at early stage (Hullár et al. 2011). Dainotti et al. (2011) construct high effective hybrid classifiers and apply a hybrid feature extraction method for early stage traffic classification. Nguyen et al. use statistical features derived from sub-flows for timely identification of VoIP traffics (Nguyen et al. 2012), and they extend the concept of early stage to "timely", since a sub-flow refers to a small number of the most recent packets taken at any point in a flow's lifetime. A. Rizzi et al. proposed a highly efficient neuro-fuzzy system for the early stage traffic identification (Rizzi et al. 2013). These machine-learning-based early identification methods all behave well for accuracy, but none of them has the ability to show and explain the selected features result when they get the well accuracy result. However, the features which can help to improve the identification performance are very important for classifier evaluation and selection. Therefore, we select FNT as the identification method, which has great ability to meet the requirement.

Flexible neural trees (FNT) (Chen et al. 2004, 2005, 2007a) is a special kind of artificial neural network proposed in recent years. The most distinctive feature is that FNT is designed with flexible tree structures. This gives FNT the ability to auto-design system structures using tree structure evolving algorithms such as immune programming (IP) (Musilek et al. 2006), probabilistic incremental program evolution (PIPE) (Salustowicz and Schmidhuber 1997) and so on. Therefore, FNT model can obtain high generalization abilities in many actual task of classification, approximation and prediction tasks (Chen et al. 2007b; Qu et al. 2008; Esposito et al. 2015; Zhou et al. 2007).

3 Proposed approach

3.1 Early stage classification

To achieve early stage traffic identification, we propose to perform classification over the most recent begin packets of

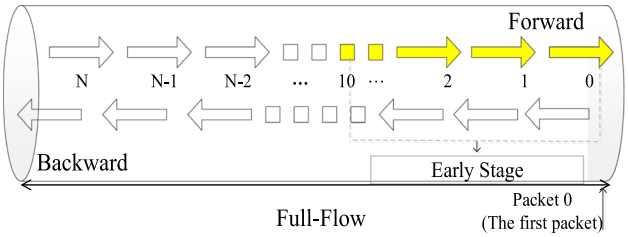

Fig. 1 Illustration of early stage of a flow

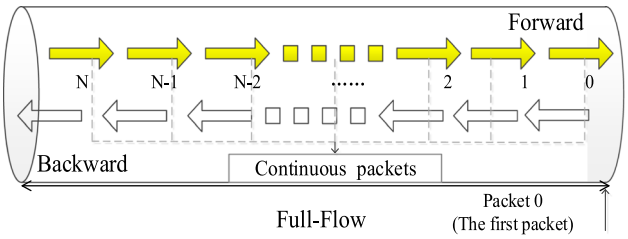

Fig. 2 Illustration of continuous early stage of a flow

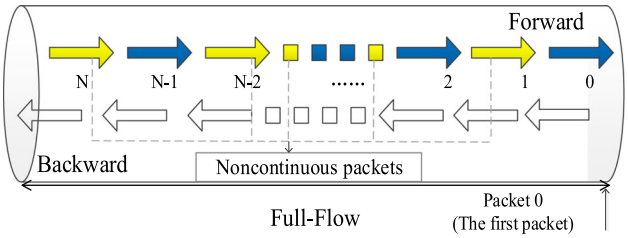

Fig. 3 Illustration of noncontinuous early stage of a flow

a bidirectional flow. In Fig. 1, we define the early stage of a traffic flow. We regard the 10 packets from the beginning of a flow as the early stage traffic packets. As the proposed research experience, the first 6 continuous packets are considered as the really early stage packets (see Fig. 2). In real world, especially in high speed core network condition, capturing continuous packets is very difficult and impractical, so we define the noncontinuous early stage of a flow (see Fig. 3). In this research, we will compare the machine learning identification efficiency between the continuous and the noncontinuous early stage.

3.2 Flexible neural tree

As introduced in Sect. 2, FNT is a special kind of artificial neural network with flexible tree structures. It is relatively easy for FNT model to obtain the near-optimal structure using tree structure optimization algorithms. For a FNT model, the leaf nodes are input nodes and the non-leaf nodes are neurons. The output of root node is the output of the whole system. FNT model uses two types of instruction sets, the function set F which is used to construct non-leaf nodes, and the terminal instruction set T for constructing nodes in tree structures. They are described

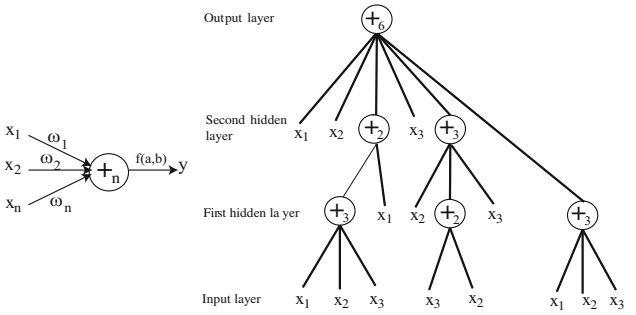

Fig. 4 A flexible neuron operator (*left*), and a typical representation of the FNT with function instruction set $F = \{+_2, +_3, +_4, +_5, +_6\}$, and terminal instruction set $T = \{x_1, x_2, x_3\}$(*right*)

as $S = F \bigcup T = \{+_2, +_3, \cdots, +_N\} \bigcup \{x_1, x_2, \ldots, x_n\}$, where $+_i (i = 2, 3, \ldots, N)$ denotes the non-leaf nodes' instructions which take i arguments. $x_1, x_2, \ldots, x_n$ are leaf nodes' instructions, and they take no other arguments. The output of a non-leaf node is calculated as a flexible neuron model (see Fig. 4). From this point of view, the instruction $+_i$ is also called a flexible neuron operator with i inputs.

In the creation process of neural tree, if a nonterminal instruction, i.e., $+_i (i = 2, 3, \ldots, N)$ is selected, i real values are randomly generated and used for representing the connection strength between the node $+_i$ and its children. In addition, two adjustable parameters a_i and b_i are randomly created as flexible activation function parameters. To develop the FNT, the following flexible activation function is used,

$$f(a_i, b_i, x) = e^{-\left(\frac{x-a_i}{b_i}\right)^2} \tag{1}$$

The total excitation function of $+_n$ is

$$\text{net}_n = \sum_{j=1}^{n} w_j \cdot x_j \tag{2}$$

where $x_j (j = 1, 2, \ldots, n)$ are the inputs to node $+_n$. The output of the node $+_n$ is calculated by,

$$\text{out}_n = f(a_i, b_i, \text{net}_n) = e^{-(\frac{\text{net}_n - a_i}{b_i})^2} \tag{3}$$

The overall output of flexible neural tree can be computed from left to right by depth-first method, recursively.

3.3 FNT tree structure optimization

PIPE Salustowicz and Schmidhuber (1997) combines probability vector coding of program instructions, population-based incremental learning (Baluja 1994), and tree-coded programs. PIPE iteratively generates successive populations of functional programs according to an adaptive probability distribution, which represented as a probabilistic prototype

tree (**PPT**), over all possible programs. Each iteration uses the best program to refine the distribution. Thus, the structures of promising individuals are learned and encoded in the **PPT**.

PPT is important for PIPE algorithm. Each program in a population is generated according to the **PPT**, which means that the **PPT** is a control factor in population generating process. And the **PPT** is also adjusted during the iterative process of the population. The **PPT** stores the knowledge gained from experiences with programs (trees) and guides the evolutionary search. It holds the probability distribution over all possible programs that can be constructed from a predefined instruction set. The **PPT** is generally a complete n-ary tree with infinitely many nodes, where n is the maximal number of function arguments.

Each node N_j in **PPT**, with $j \geq 0$, contains a variable probability vector $\overrightarrow{P_j}$. Each $\overrightarrow{P_j}$ has n components, where n is the number of instructions in instruction set S. Each component $P_j(I)$ of $\overrightarrow{P_j}$ denotes the probability of choosing instruction $I \in S$ at node N_j. Each vector $\overrightarrow{P_j}$ is initialized as follows:

$$P_j(I) = \frac{P_T}{l} \quad \forall I : I \in T \tag{4}$$

$$P_j(I) = \frac{1 - P_T}{k} \quad \forall I : I \in F \tag{5}$$

where l is the number of instructions in set T, and k is the number of instructions in set F. P_T is the total probability to select terminal instructions.

PIPE combines two forms of learning: generation-based learning (GBL) and elitist learning (EL). GBL is a learning strategy according to the best program of current population, and EL is a learning strategy according to the global best program which is also called the elitist program. GBL is PIPE's main learning algorithm. The purpose of EL is to use the best program found so far as an attractor. The whole PIPE learning frame is as follows:

1: *repeat*
2: with probability P_{el} *do* EL
3: otherwise *do* GBL
4: *until* termination criterion is reached

Here, P_{el} is a user-defined constant in [0, 1].

3.3.1 Generation-based learning

The main steps of the generation-based learning can be described as follows:

Step 1. Creation of program population. A population of programs P^j_{rog} ($0 < j \leq PS$; PS is population size) are generated using the prototype tree **PPT**.

Step 2. Population evaluation. Each program P^j_{rog} of the current population is evaluated on the given task, and a fitness value $F(P^j_{\text{rog}})$ is assigned according to the fitness function. The best program of the current population (the one with the smallest fitness value) is denoted as P^b_{rog} . The best program found so far (elitist) is preserved in P^{el}_{rog}.

Step 3. Learning from population. Prototype tree probabilities are modified such that the probability $P(P^b_{\text{rog}})$ of creating P^b_{rog} increases. This procedure is called adapting **PPT** towards P^b_{rog}. This is implemented as follows. First $P(P^b_{\text{rog}})$ is computed by looking at all **PPT** nodes N_j used to generate P^b_{rog}:

$$P(P^b_{\text{rog}}) = \prod_{j:N_j \text{ used to generate } P^b_{\text{rog}}} P_j(I_j(P^b_{\text{rog}})) \tag{6}$$

where $I_j(P^b_{\text{rog}})$ denotes the instruction of program P^b_{rog} at node position j. Then, a target probability P_{tar} for P^b_{rog} is calculated:

$$P_{\text{tar}} = P(P^b_{\text{rog}}) + (1 - P(P^b_{\text{rog}})) \cdot lr \cdot \frac{\varepsilon + F(P^{el}_{\text{rog}})}{\varepsilon + F(P^b_{\text{rog}})} \tag{7}$$

Here, lr is a constant learning rate, and ε is a positive user-defined constant. Given P_{tar}, all single node probabilities $P_j(I_j(P^b_{\text{rog}}))$ are increased iteratively:
do

$$P_j(I_j(P^b_{\text{rog}})) = P_j(I_j(P^b_{\text{rog}})) + c \cdot lr \cdot (1 - P_j(I_j(P^b_{\text{rog}}))) \tag{8}$$

until $P_j(I_j(P^b_{\text{rog}})) \geq P_{\text{tar}}$
where c is a constant influencing the number of iterations. The smaller c, the higher the approximation precision of P_{tar} and the number of the required iterations. Setting $c = 0.1$ turns out to be a good compromise between precision and speed. And then all adapted vectors $\overrightarrow{P_j}$ are renormalized.

Step 4. Mutation of prototype tree. All probabilities $P_j(I)$ stored in nodes N_j that are accessed to generate program P^b_{rog} are mutated with probability P_{M_p} :

$$P_j(I) = P_j(I) + mr \cdot (1 - P_j(I)) \tag{9}$$

where mr is the mutation rate, another user-defined parameter. Also all mutated vectors $\overrightarrow{P_j}$ are renormalized.

Step 5. Prototype tree pruning. At the end of each generation, the prototype tree is pruned. **PPT** subtrees attached to nodes that contain at least one probability vector component above a threshold T_P can be pruned.

Step 6. Termination criteria. Repeat the above procedure until a fixed number of program evaluations is reached or a

satisfactory solution is found. In our study, we use two rules to terminate the iteration: one is the maximum iteration number, the other is critical fitness. Either the iteration number has reached the maximum value or the fitness of the global best program has achieved the critical value, the iterative process will be terminated.

3.3.2 Elitist learning

Elitist learning, whose basic flow is the same as that of GBL, focuses search on previously discovered promising parts of the search space. In GBL, the **PPT** is adapted towards the best program of the current population. However, in EL, the **PPT** is adapted towards the elitist program. So we also use Equation (8) and (9) to adapt the **PPT**, but P^{b}_{rog} is replaced with P^{el}_{rog} in these equations. EL is particularly useful with small population sizes and works efficiently in the case of noise-free problems. To learn the structure and parameters of a FNT simultaneously, there is a trade-off between the structure optimization and the parameter learning. In fact, if the structure of the evolved model is not appropriate, it is not useful to pay much attention to the parameter optimization. On the contrary, if the best structure has been found, the further structure optimization may destroy the best structure. In this paper, a technique for balancing the structure optimization and the parameter learning is used. If a better structure is found, then it does local search (simulated annealing) for a number of steps (maximum allowed steps), or it stops in case no better parameter vector will be found for a significantly long time (say 100 to 2000 in our experiments). The criterion of the better structure is distinguished as follows: if the fitness value of the best program is smaller than the fitness value of the elitist program, or the fitness values of the two programs are equal but the nodes of the former are lower than the later, then we say that a better structure is found.

3.4 FNT parameter optimization

Probabilistic Incremental Program Evolution(PIPE) algorithms and Particle swarm optimization (PSO) algorithms are used for evolution of the tree structure and tree parameters. PSO (Kennedy 2010; Yoshida et al. 2000) conducts searches using a population of particles that correspond to individuals in an evolutionary algorithm. A population of particles are randomly generated initially. Each particle represents a potential solution, and it has a position represented by a position vector $\mathbf{x}_i$. A swarm of particles move through the problem space, with the moving velocity of each particle represented by a velocity vector $\mathbf{v}_i$. At each time step, a function f_i representing a quality measure is calculated using $\mathbf{x}_i$ as input. Each particle keeps track of its own best position, which is associated with the best fitness it has achieved so far in a vector $\mathbf{p}_i$. Furthermore, the best position among all the particles obtained so far in the population is kept track as $\mathbf{p}_g$. The best position means the best global solution obtained so far. In addition to this global version, another version of PSO keeps track of the best position among all the topological neighbors of a particle (For more information on neighborhood topology, we refer to Kennedy 1999; Krink et al. 2002).

At each time step t, using the individual best position $\mathbf{p}_i$ and the global best position $\mathbf{p}_g(t)$, a new velocity for particle i is updated by

$$\mathbf{v}_i(t+1) = \mathbf{v}_i(t) + c_1\phi_1(p_i(t) - \mathbf{x}_i(t)) + c_2\phi_2(p_g(t) - \mathbf{x}_i(t)) \tag{10}$$

where c_1 and c_2 are positive constants and ϕ_1 and ϕ_2 are uniformly distributed random numbers in [0,1]. The term $\mathbf{v}_i$ is limited to the range of $\pm\mathbf{v}_{\max}$. If the velocity violates this limit, it is set to its proper limit. This method of changing velocity enables the particle i to search around its individual best position $\mathbf{p}_i$, and global best position $\mathbf{p}_g$. Based on the updated velocities, each particle changes its position according to the following equation:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{11}$$

Based on Eqs. (10) and (11), the population of particles tend to cluster together with each particle moving in a random direction. This may result in premature convergence in many problems. An effective method to avoid premature convergence is to update the velocity as the following formula (Kennedy 1999):

$$\begin{aligned}\mathbf{v}_i(t+1) = \chi(&\omega\mathbf{v}_i(t) + c_1\phi_1(p_i(t) - \mathbf{x}_i(t)) \\ &+ c_2\phi_2(p_g(t) - \mathbf{x}_i(t)))\end{aligned} \tag{12}$$

where two new parameters, χ and ω, are also real numbers. The parameter χ controls the magnitude of $\mathbf{v}$, whereas the inertia weight ω weights the magnitude of the old velocity $\mathbf{v}_i(t)$ in the calculation of the new velocity $\mathbf{v}_i(t+1)$.

4 Experiment approach

4.1 Data traces

4.1.1 Auckland II traffic traces

Auckland II is a collection of long GPS-synchronized traces which are taken using a pair of DAG 2 cards at the University of Auckland which is available at (WAND 2009). There are 85 trace files which were captured from November 1999 to July 2000. Most traces were targeted at 24 h runs,

Table 1 Characteristics of Auckland II traces

Type	#instances	Total bytes
ftp	251	136241
ftp-data	463	5260804
http	23721	139421961
imap	193	86455
pop3	498	98699
smtp	2602	1230528
ssh	237	149502
telnet	37	21171

Table 2 Characteristics of UNIBS traces

Type	#instances	Total bytes
bittorrent	3571	6393487
edonkey	379	241587
http	25729	107342346
imap	327	860226
pop3	2473	4292419
skype	801	805453
smtp	120	43566
ssh	23	39456

but hardware failures have resulted in most traces being significantly shorter. We select two trace files captured at Feb 14 2000 (20000214-185536-0.pcap and 20000214-185536-1.pcap) for our study. The traces include only the header bytes, with a maximum amount of 64 bytes for each frame, while the application payload is fully removed. And all IP addresses are anonymised using Crypto-Pan AES encryption. The header traces were captured with a GPS-synchronized mechanism using a DAG3.2E card connected to a 100 Mbps Ethernet hub interconnecting the University's firewall to their border router.

Since the application payloads were not recorded in Auckland II, DPI tools are invalid to obtain ground truths. The only way to pick out the original application type using port numbers. In this study, we only accounted the TCP case since TCP is the predominant transport layer protocol. Each flow is thus assigned to the class identified by the server port. We select eight main types from Auckland II traces and filtered mouse flows with no more than six non-zero packets as illustrated in Sect. 3. Table 1 lists all selected types and their instance and total bytes distributions.

4.1.2 UNIBS traffic traces

UNIBS is another opening traffic traces developed by Prof. F. Gringoli and his research team, which is available at (NTW 2009). They developed a useful system namely GT (Gringoli et al. 2009) for the application ground truths of the captured Internet traffics. The traces were collected on the edge router of the campus network of the University of Brescia on three consecutive working days (Sept 30, Oct 1 and Oct 2 2009). They are composed of traffic generated by a set of twenty workstations running the GT client daemon. Traffics were collected by running Tcpdump (Jacobsen et al. 2005) on the Faculty's router, which is a dual Xeon Linux box that connects the network to the Internet through a dedicated 100 Mb/s uplink. 99 % flows in UNIBS are TCP flows. Therefore, we again use TCP flows in this data set for our study. Using GT, UNIBS traces recorded the application information of each captured flow. We can get the application ground truths by both TCP port numbers and GT records. We also chose eight main types in UNIBS for our study which are shown in Table 2. Different from Auckland II traces, there are two popular P2P applications in this data set, bittorrent and edonkey, which are recorded by GT. Skype is also selected as an import Internet application. Flows with no more than six non-zero payload packets are also filtered. The instance and total byte distributions of each type are listed in Table 2.

4.1.3 UJN traffic traces

The third data set is collected in a laboratory network of University of Jinan using Traffic Labeler (TL) (Peng et al. 2014b). TL system captures all user socket calls and their corresponding application process information in the user mode on a Windows host, and it sends the information to an intermediate NDIS driver in the kernel mode. The intermediate driver writes application type information on the TOS field of the IP packets which match the 5-tuple. By this mean, each IP packet sent from the Windows host carries their application information. Therefore, traffic samples collected on the network have been labeled with the accurate application information, and they can be used for training effective traffic classification models. We deployed 10 TL instances on Windows user hosts in the laboratory network of Provincial Key Laboratory for Network Based Intelligent Computing. A mirror port of the uplink port of the switch was set, and a data collector was deployed at the mirror port. The deployed TL instances ran at work hours every day. The data collecting process lasted 2 days in May 2013. Again, flows with no more than six non-zero payload packets are also filtered. Table 3 shows the instance and the total byte distributions of each type.

4.2 Early stage identification features

- *Packet size* Packet size is proved to be the most effective original packet-level feature in the early stage of traffics

Table 3 Characteristics of UJN traces

Type	#samples	Total bytes
Web browser	11890	58025350
Chat	11478	60212804
Cloud disk	1563	109552924
Live update	2169	28759962
Stream media	810	785556
Mail	803	2092862
P2P	326	2521089
Other	1408	3635558

(Este et al. 2009a). We use the packet sizes of the first six packets as the original early stage traffic features in this study, since we have proven that the first six packets are most effective for the early stage feature extraction (Peng et al. 2014a). And all statistical features are computed based on the packet sizes.

- *Average* The average is also known as the arithmetical mean, which is an extensively used statistical indicator. This feature is calculated as follows:

$$\text{avg} = \sum_{i=1}^{n} ps_i \tag{13}$$

- *Standard deviation* The standard deviation shows how much variation or dispersion from the average exists. And the feature is defined as:

$$\text{stdev} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(ps_i - \text{avg})^2} \tag{14}$$

where n is the number of packets, i. e. 6 in this study.

- *Maximum and minimum* The maximum and the minimum payload size are also applied in the study, and we use the abbreviations of max and min, respectively.

4.3 Compared methods

We execute our identification experiments using six well-known machine learning classifiers. We use Weka data mining software (Waikato 2013) as our experiment tool. All classifiers are run in Weka and all processed data sets are formatted into the Weka data file with the extension name of "arff". The classifiers we selected fall into five categories according to Weka:

- *Bayes* Bayes classifiers are based on Bayes theorem, which is widely applied in many engineering areas. In this study, we choose Bayesian network (BayesNet) (Friedman et al. 1997) as Bayes classifiers.
- *Meta* Strictly speaking, meta classifier is a kind of classification framework based on a specific classifier. This technique first trains a group of "weak learn", and then it generates a "strong learn" based on the weak learns. We choose Bagging (Breiman 1996) for our study.
- *Rule* As the name suggests, a rule based classifier extracts rules using a specific policy, e. g. probability and decision trees, and it uses the rules to classify testing data. OneR (Holte 1993) and PART (Frank and Witten 1998) are selected for this category in this study.
- *Trees* This refers to decision trees. A decision tree divides the target feature space hierarchically. Each division produces a node on the decision trees. A classification procedure is a procedure that goes from the root node to a specific leaf node on the tree. In this study, Naive Bayesian trees (NBTree) (Kohavi 1996) and random forest (RandomForest) (Svetnik et al. 2003) are selected for this category.
- *Functions* Weka refers all classifiers based on specific functions to this category. We choose support vector machine (SVM) (Vapnik and Vapnik 1998) and radial basis function neural network (RBFNetwork) (Broomhead and Lowe 1988) for this category.

Table 4 lists all classifiers applied in this study. We cite the original literature of each classifier in the table. Readers can find technical details of each classifier in its corresponding literature.

4.4 Accuracy performance

The confusion matrix is the basis in measuring a classification task, wherein rows denote the actual class of the instances, and the columns denote the predicted class. Figure 5 shows a typical confusion matrix of a binary classification. The simplest method to evaluate a classifier is using the classification accuracy (acc) which is defined as the rate between the number of samples correctly classified and the total number of samples in testing set. But the accuracy can only express the

Table 4 Classifiers selected in the study

Classifiers	Type
BayesNet (Friedman et al. 1997)	Bayes
Bagging (Breiman 1996)	Meta
OneR (Holte 1993)	Rule
PART (Frank and Witten 1998)	Rule
NBTree (Kohavi 1996)	Trees
RandomForest (Svetnik et al. 2003)	Trees
SVM (Vapnik and Vapnik 1998)	Functions
RBFNetwork (Broomhead and Lowe 1988)	Functions

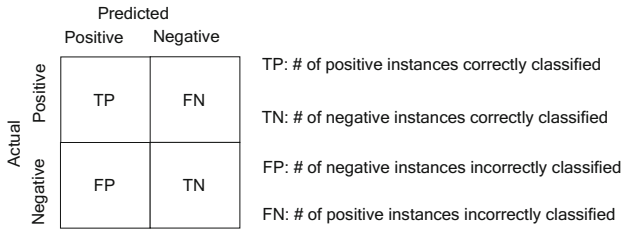

Fig. 5 Confusion Matrix

overall level of hitting ratio, and it does not contain particular information of incorrectly classified sample ratio of each class. So a more sophisticated and widely used evaluating method is applied in this research. This method uses True Positive Rate (TPR) and False Positive Rate (FPR) to evaluate the performance of a classifier. TPR and FPR are deduced from the confusion matrix as Fig. 5 shows. For each class, a confusion matrix can be obtained according to the classification results. And then its TPR and FPR are defined as:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{15}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{TN} + \mathrm{FP}} \tag{16}$$

It can be seen from these two equations that TPR is in fact the ratio of the correctly classified positive samples and the total positive samples, and the FPR is the ratio of the incorrectly classified negative samples and the total negative samples. It can be inferred easily that:

$$\mathrm{acc} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{17}$$

5 Results and analysis

5.1 Results

Figure 6 shows the experimental results of the Auckland II data set using the first 6 packets. First of all, FNT hits the highest TRP and the lowest FPR values in all of the compared methods, and it performs perfectly in this case. The tree classifiers, including NBTree and RandomForest, show good performances in the experiments. In fact, FNT is also a special kind of tree classifier. Bagging and PART also behave fairly well. SVM accidentally gets the highest FPR value, and RBFNetwork also does not perform very well.

We select 6 noncontinuous packets from the first 10 packets of the Auckland II data set for the next group of experiments. The selected packet numbers are 1, 4, 6, 7, 9, 10, and the noncontinuous packet numbers are also used for the other two data sets. Figure 7 gives the experimental results

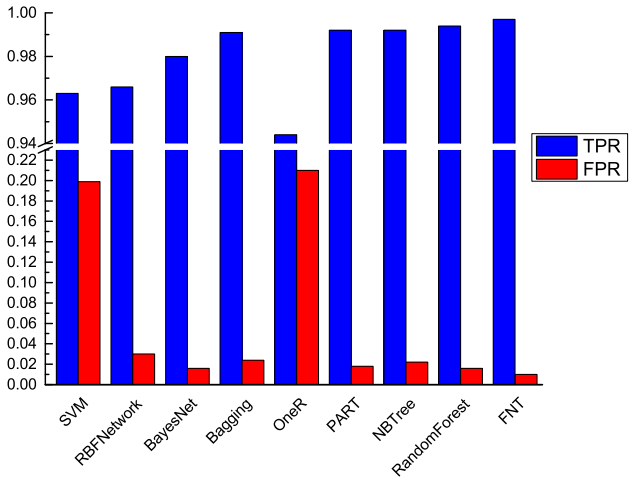

Fig. 6 Results of Auckland II data set using the first 6 packets

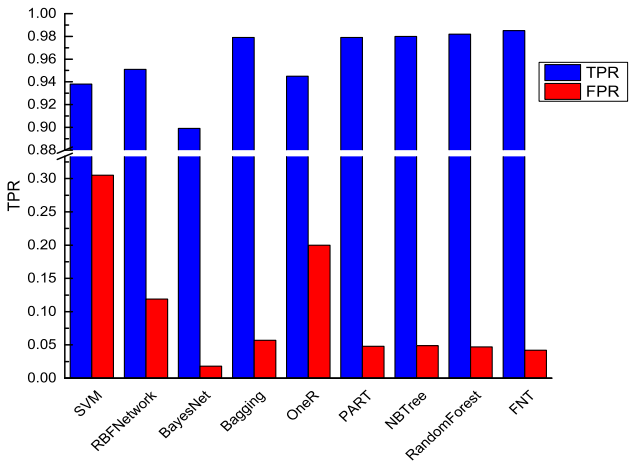

Fig. 7 Results of Auckland II data set using noncontinuous 6 packets

of the Auckland II data set using the noncontinuous 6 packets. FNT gets the highest TPR value again, and its FPR value is also very low. The lowest FPR value is hit by BayesNet this time; however, its TPR value is also the lowest one. Bagging, PART, NBTree and RandomForest gain stable and high performances again.

When comparing the two groups of results of the Auckland II data set, the whole performances of the noncontinuous 6 packets are not as well as that of the first 6 packets. Most of the classifiers get higher TPR and lower FPR values for the later case. The results show that a continuous packet sequence is more effective than a noncontinuous one for the Auckland II data set.

The two groups of experimental results for the UNIBS data set are shown in Figs. 8 and 9. FNT gets high performances again for this data set: the second highest TPR and lowest FPR values using the first 6 packets, and the highest TPR and the second lowest FPR values using the noncontinuous 6 packets. Bagging, PART, NBTree and RandomForest again

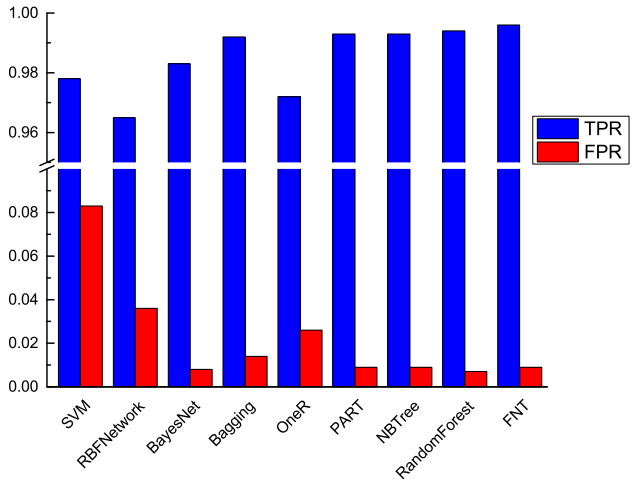

Fig. 8 Results of UNIBS data set using the first 6 packets

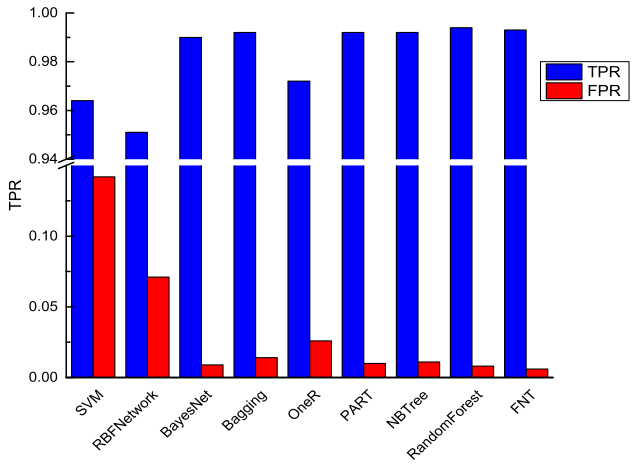

Fig. 9 Results of UNIBS data set using noncontinuous 6 packets

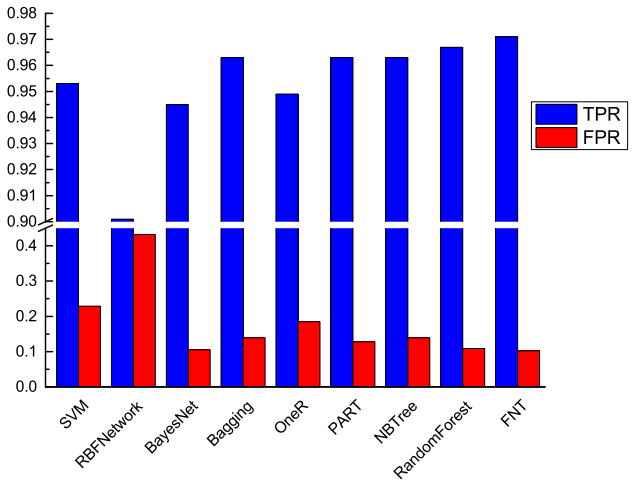

Fig. 10 Results of UJN data set using the first 6 packets

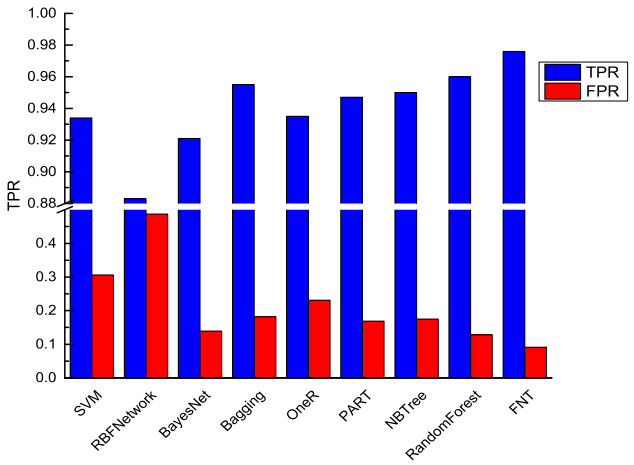

Fig. 11 Results of UJN data set using noncontinuous 6 packets

get high TPR and low FPR values from a global view. SVM and RBFNetwork get high FPR values, which means that the two function classifiers misclassified many traffic instances. It can be observed from Figs. 8 and 9 that most of the selected classifiers get higher performances using the first 6 packets than using the noncontinuous 6 packets. This is in accordance with that of the Auckland II data set.

Figures 10 and 11 show the experimental results of the UJN data set. FNT outputs the highest TPR and the lowest FPR values for both of the first 6 packets and the noncontinuous 6 packets cases. The results have shown again the effectiveness of FNT model for traffic identification. The relatively high TPR and low FPR values show once again the stable and high performances of Bagging, PART, NBTree and RandomForest. In contrast, SVM and RBFNetwork still do not get good performances.

The comparison between Figs. 10 and 11 can testify again the advantages of the method using the first 6 packets: for most classifiers, the TPR values are relatively higher and the FPR values are relatively lower than that of the noncontinuous 6 packets cases.

Then, FNT classifier trains itself by the labeled and normalized early packet size data set. The selected data records of the first 6 packets size($_{\text{size0,size1,size2,size3,size4,size5}}$), Average size($_{\text{size6}}$) of the first 6 packets, Standard deviation size($_{\text{size7}}$) of the first 6 packets, Maximum size($_{\text{size8}}$)and Minimum size($_{\text{size9}}$) of the first 6 packets were used as the inputs (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) of FNT. For identifying continuous packets in UJN data set as an example, after the optimization of the tree structure by PIPE algorithms and optimization of tree weights by PSO algorithms, FNT performance is illustrated in Fig. 12. As can be learnt from the optimized FNT tree result, except the first packet size, the second packet size and the minimum of the first 6 packets size, all other features are selected as effective features for the early stage IP traffic identification. Actually, the first packet and the second packet are packets generated by the connection and verification activity. It is useless for identifying different applications. Generally, the minimum packets

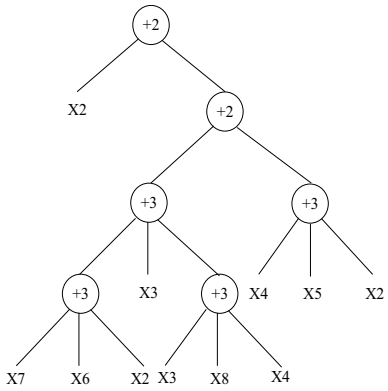

Fig. 12 The optimized FNT tree result

are the end of a flow, so it is also useless for differing the application. From the tree result and analysis of Internet and IP traffic packet character, FNT model really has a good ability to select useful variables (features) for the classification.

5.2 Analysis and discussions

Our experiments gained many interesting things, and we will discuss them as follows.

- First of all, FNT shows its high performances in the experiments, which has proven the effectiveness of FNT for traffic identification. We selected three traffic data sets in the experiments, and FNT got the highest TPR values and the lowest FPR values for most of the cases. The results mean that FNT is able to get the highest identification rates and the lowest misclassification rates for traffic identification problems.
- Second, the experimental results show that a continuous packet sequence is more effective than a noncontinuous one. Most of the selected classifiers output higher performances when using the first 6 packet for all of the three data sets. It is not hard to be comprehend that a continuous packet sequence can contain more complete feature information than a noncontinuous packet sequence.
- Finally, although a noncontinuous packet sequence is less effective than a continuous packet for identification result, it still has acceptable efficiency, which can satisfied the identification requirement in real-world network condition. Furthermore, the selected effective features can help to optimize the training and identification result.

6 The real-world implementation structure

For IP traffic measuring, an implementable classification system is very important. In this section, we designed an implementable ML-based early stage traffic identification structure (Fig. 13), which can be implemented to a LAN to measure the traffic in real world. The scheme includes four main components of TL system, Programable gateway, ML training server and ML-based identification server. Experiment platform according to the designed structure was implemented in the Network research laboratory of University of Jinan.

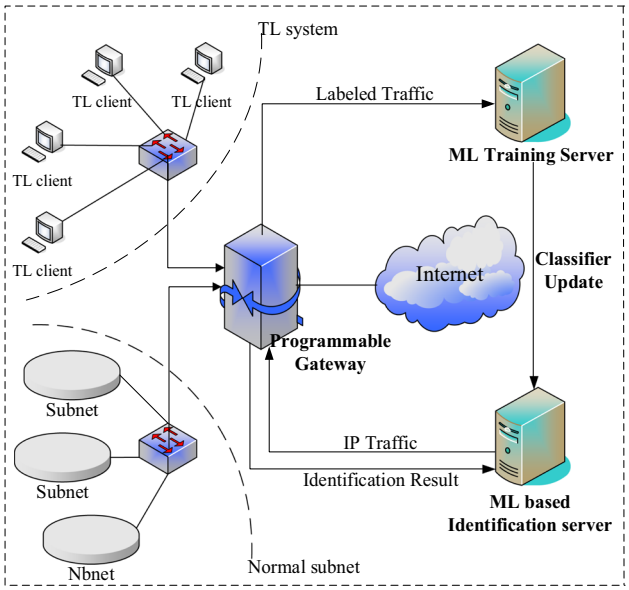

Fig. 13 The real-world implementation structure

(1) TL system. We have designed a model named Traffic Labeller (TL) system. This system captures all user socket calls and their corresponding application process information in the user mode on a Windows host. Once a sending data call has been captured, its 5-tuple source IP, destination IP, source port, destination port and transport layer protocol, associated with its application information, is sent to an intermediate NDIS driver in the kernel mode. Then, the intermediate driver writes application type information on TOS field of the IP packets which match the 5-tuple. In this way, each IP packet sent from the Windows host carries their application information. Therefore, traffic samples collected on the network have been labeled with the accurate application information, and it can be used for training effective traffic classification models.

(2) Programable gateway. As can be learnt from Fig. 13, the traffic from TL system network area and normal network area will be forwarded to the Internet through the network gateway. We have designed an online hybrid traffic classifier for Peer-to-Peer systems based on network processors (Chen et al. 2009), a famous programable hardware. In this research, we use the network processors as the hardware platform of programable network gateway. Before forwarding the IP traffic packet, it will check the TOS field of an IP packets. The

packet will be mirrored to the ML training server if a application label is found, otherwise it will be mirrored to the ML-based identification server.

(3) ML training server. When the ML training server captures the labeled IP traffic, the early stage features will be extracted. In our implementation, we do not learn an identification model online. Instead, we train the Machine Leaning model such as FNT offline on the serve, and then we transfer or update the learned model to the ML-based identification server for identifying IP traffic as early as possible.

(4) ML-based identification server. Once the classifier has been trained on ML training server, it can be updated to the ML-based identification server. The traffic from the normal network can be identified based on their early stage features. The system manager can monitor the classifier according to the requirements, it can add, delete or modify the classification knowledge to ensure the classification accuracy, and reduce the classification FPR (False Negative Rate), and then enhance the TPR (True Positive Rate) to meet the real-world identification requirement.

7 Conclusion and future work

In this paper, we have tried to build an effective early stage traffic identification model using Flexible Neural Networks. We use three traffic data sets including two opening data sets for the experimental evaluations. And eight classical classification algorithms are applied for experimental comparisons. According to the experimental results, we conclude that FNT is effective for the early stage traffic identification. As can be seen from the experimental results that FNT outperforms the other six classifiers for most experiment cases. Furthermore, FNT does not only get high total identification rates, but also shows good trade-off among different traffic types. And this is very important for traffic identification, especially for the cases with an imbalanced data distribution.

It is often a difficult task to select variables (features) for the classification problem, especially when the feature space is large. A fully connected NN classifier usually cannot do this. In the perspective of FNT framework, the nature of model construction procedure allows the FNT to identify important input features in building a traffic classifier that is computationally efficient and effective. In the future, we plan to design a distributed hierarchical early stage classification structure, which can make full use of the selected optimal feature result feedback to help improve the other distributed classifiers.

Acknowledgements This study is supported by the National Natural Science Foundation of China under Grant No. 61472164, the Natural Science Foundation of Shandong Province under Grant Nos. ZR2014JL042 and ZR2012FM010.

Compliance with ethical standards

Conflict of interest The authors declared that they have no conflicts of interest to this work.

References

Baluja S (1994) Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning. Tech report, Carnegie Mellon University

Bernaille L, Teixeira R, Akodkenou I, Soule A, Salamatian K (2006) Traffic classification on the fly. ACM SIGCOMM Comput Commun Rev 36(2):23–26

Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140

Broomhead DS, Lowe D (1988) Multivariable functional interpolation and adaptive networks. Proc Ijcai 2(3):321–355

Chen Y, Yang B, Dong J (2004) Nonlinear system modelling via optimal design of neural trees. Int J Neural Syst 14(02):125–137

Chen Y, Yang B, Dong J, Abraham A (2005) Time-series forecasting using flexible neural tree model. Inf Sci 174(3):219–235

Chen Y, Chen F, Yang JY (2007a) Evolving mimo flexible neural trees for nonlinear system identification. In: IC-AI2007, CSREA Press pp 373–377

Chen Y, Yang B, Abraham A (2007b) Flexible neural trees ensemble for stock index modeling. Neurocomputing 70(4):697–703

Chen Z, Yang B, Chen Y, Abraham A, Grosan C, Peng L (2009) Online hybrid traffic classifier for peer-to-peer systems based on network processors. Appl Soft Comput 9:685–694

Dainotti A, Pescapé A, Sansone C (2011) Early classification of network traffic through multi-classification. Springer, New York, pp 122–135

Dainotti A, Pescape A, Claffy KC (2012) Issues and future directions in traffic classification. Netw IEEE 26(1):35–40

Dainottia A, Pescap A, Rossib PS, Palmieric F, Ventrea G (2008) Internet traffic modeling by means of hidden markov models. Comput Netw 52:2645–2662

Esposito C, Ficco M, Palmieri F, Castiglione A (2013) Interconnecting federated clouds by using publish-subscribe service. Clust Comput 16:887–903

Esposito C, Ficco M, Palmieri F, Castiglione A (2015) Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory. IEEE Trans Comput. doi:10.1109/TC.2015.2389952

Este A, Gringoli F, Salgarelli L (2009a) On the stability of the information carried by traffic flow features at the packet level. ACM SIGCOMM Comput Commun Rev 39(3):13–18

Este A, Gringoli F, Salgarelli L (2009b) Support vector machines for tcp traffic classification. Comput Netw Int J Comput Telecommun Netw 53(14):2476–2490

Frank E, Witten IH (1998) Generating accurate rule sets without global optimization. In: Morgan Kaufmann (ed) Proceeding of International Conference on Machine Learning, pp 144–151

Friedman N, Geiger D, Goldszmidt M (1997) Bayesian network classifiers. Mach Learn 29(2–3):131–163

Gringoli F, Salgarelli L, Dusi M, Cascarano N, Risso F et al (2009) Gt: picking up the truth from the ground for internet traffic. ACM SIGCOMM Comput Commun Rev 39(5):12–18

Holte RC (1993) Very simple classification rules perform well on most commonly used datasets. Mach Learning 11(1):63–90

Huang NF, Jai GY, Chao HC (2008) Early identifying application traffic with application characteristics. In: Communications, ICC'08. IEEE international conference on, IEEE, pp 5788–5792

Huang NF, Jai GY, Chao HC, Tzang YJ, Chang HY (2013) Application traffic classification at the early stage by characterizing application rounds. Inf Sci 232:130–142

Hullár B, Laki S, György A (2011) Early identification of peer-to-peer traffic. In: Communications (ICC), 2011 IEEE international conference on, IEEE, pp 1–6

Jacobsen V, Leres C, McCanne S (2005) Tcpdump/libpcap. http://www.tcpdump.org

Kennedy J (1999) Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In: Evolutionary computation, CEC 99, proceedings of the 1999 congress on, IEEE, vol 3

Kennedy J (2010) Particle swarm optimization. Swarm Intell 1(1):33–57

Kohavi R (1996) Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In: KDD'96, AAAI Press, pp 202–207

Krink T, VesterstrOm JS, Riget J (2002) Particle swarm optimisation with spatial particle extension. In: CEC'02, IEEE, pp 1474–1479

Li W, Moore AW (2007) A machine learning approach for efficient traffic classification. In: Modeling, analysis, and simulation of computer and telecommunication systems. MASCOTS'07. 15th international symposium on, IEEE, pp 310–317

Moore A, Zuev D, Crogan M (2005) Discriminators for use in flow-based classification. Tech report, Queen Mary and Westfield College

Moore AW, Zuev D (2005) Internet traffic classification using bayesian analysis techniques. ACM SIGMETRICS Perform Eval Rev ACM 33:50–60

Musilek P, Lau A, Reformat M, Wyard-Scott L (2006) Immune programming. Inf Sci 176(8):972–1002

Nguyen TT, Armitage G, Branch P, Zander S (2012) Timely and continuous machine-learning-based classification for interactive ip traffic. IEEE/ACM Trans Netw (TON) 20(6):1880–1894

NTW (2009) Unibs: data sharing. http://www.ing.unibs.it/ntw/tools/traces/

Palmieri F, Fiore U (2009) A nonlinear, recurrence-based approach to traffic classification. Comput Netw 53:761–773

Peng L, Yang B, Chen Y, Wu T (2014a) How many packets are most effective for early stage traffic identification: an experimental study. Commun Chin 11(9):183–193

Peng L, Zhang H, Yang B, Chen Y, Wu T (2014b) Traffic labeller: collecting internet traffic samples with accurate application information. Commun Chin 11(1):69–78

Qu B, Zhang Z, Guo L, Meng D (2012) On accuracy of early traffic classification. In: Networking, architecture and storage (NAS), 2012 IEEE 7th international conference on, IEEE, pp 348–354

Qu SN, Liu Zl, Cui G, Zhang B, Wang S (2008) Modeling of cement decomposing furnace production process based on flexible neural tree. In: Information management, innovation management and industrial engineering, ICIII'08. international conference on, IEEE, vol 3, pp 128–133

Rizzi A, Colabrese S, Baiocchi A (2013) Low complexity, high performance neuro-fuzzy system for internet traffic flows early classification. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th international, IEEE, pp 77–82

Salustowicz R, Schmidhuber J (1997) Probabilistic incremental program evolution. Evol Comput 5(2):123–141

Svetnik V, Liaw A, Tong C, Culberson JC, Sheridan RP, Feuston BP (2003) Random forest: a classification and regression tool for compound classification and qsar modeling. J Chem Inf Comput Sci 43(6):1947–1958

Vapnik VN, Vapnik V (1998) Statistical learning theory, vol 1. Wiley, New York

Waikato U (2013) Weka 3: data mining software in java. http://www.cs.waikato.ac.nz/ml/weka/

WAND (2009) Wits: Waikato internet traffic storage. http://www.wand.net.nz/wits

Yoshida H, Kawata K, Fukuyama Y, Takayama S, Nakanishi Y (2000) A particle swarm optimization for reactive power and voltage control considering voltage security assessment. Power Syst IEEE Trans 15(4):1232–1239

Zhang J, Chen X, Xiang Y, Wu J (2014) Robust network traffic classification. IEEE/ACM Trans Netw 24:84–88

Zhou J, Liu Y, Chen Y (2007) Ica based on kpca and hybrid flexible neural tree to face recognition. In: Computer information systems and industrial management applications, CISIM'07. 6th international conference on, IEEE, pp 245–250