Journal of Applied Mathematics and Physics, 2013, 1, 56-62
http://dx.doi.org/10.4236/jamp.2013.14011 Published Online October 2013 (http://www.scirp.org/journal/jamp)

EPFIA: Extensible P2P Flows Identification Architecture

Bo Xu, Bing Li, Chao Hu, Guomin Zhang
Institute of Command Information System PLA University of Science and Technology, Nanjing, China
Email: xubo820@163.com, Libing781@163.com, Huchao1007@163.com, zhang_gmwn@163.com

Received August 2013

ABSTRACT

The fundament of managing P2P traffic is identifying various P2P flows accurately. Although many P2P flows identification methods are presented nowadays, there are no ideas for either integrating these independent methods together or being extended fast to support new method. In this work, an extensible P2P flows identification architecture (EPFIA for short) is proposed. In order to identify many specific P2P flows, EPFIA uses many different identification methods simultaneously, and obtains the highest efficiency via adjusting their identification sequence. An online mechanism of renewing identification methods is designed, which can extend new identification method without compiling the whole program. Applying policy mechanism, identification methods can be updated, started and halted remotely. The experiment results of running the prototype system show us that EPFIA could effectively promote the performance of system and support online renew P2P identification methods and manage them remotely.

Keywords: P2P; Flow Identification; Architecture; Policy Schedule

1. Introduction

P2P technology is one of the most exciting areas of Internet, which uses the idle resources in network edge, and makes Internet computing pattern from centralized to edge. Recent studies show that more than 60% of Internet traffic is attributed to P2P applications, of which BitTorrent and eDonkey traffic is of 80% of the total P2P traffic [1].

In this paper we define flow as bidirectional flow determined by 5-tuple (source and destination address, source and destination port, and transport layer protocol) and 64 seconds timeout is adopted [2]. Generally P2P traffic identification methods can be divided into 4 types: transport layer port based identification method (f_{port} for short), application layer signature based identification method (f_{sig} for short), transport layer behavior based heuristic identification method ($f_{behavior}$ for short), and machine learning based identification method (f_{ML} for short). After the invalidation of f_{port} [3], researchers pay more attention to f_{sig}, which uses the packet payload signature to identify the application accurately [3-6]. $f_{behavior}$ uses flow attributes, statistics and behavior feature to identify P2P application[7,8]. f_{ML} constructs classifier with the flow characteristics of different protocol, and then use the classifier to identify the applications of unknown flows [9,10]. Besides, there are also crawlers based identification methods, etc. As the development of new P2P applications, new identification methods will keep on emerging. Thus, current P2P flow identification methods have the following problems: firstly, identifying one kind of P2P applications need one kind of device, this will be too hard to deploy system that can identify many kinds of P2P applications; secondly, the structure of identification system is inflexible, and it needs recompile the system software (even redesign system architecture) to extend or modify some new identification methods; thirdly, updating or maintaining identification system is very difficult.

We propose EPFIA to assemble many P2P traffic identification methods. EPFIA has the following features: improving identification efficiency via optimizing the sequence of identifying different P2P applications; designing a mechanism to update identification methods online, which can extend the P2P flow identification function without recompile the software; using policy mechanism, the update, start, and stop identification method can be carried out remotely.

Moore and Papagiannaki used port numbers and application layer payload to identify P2P flow [6]. They sorted the nine identification methods based on the ascending order of implementation complexity so as to identify the application of each flow, and pay no attention to the impact of the component of traffic on identification system performance. Most of current works focus on P2P application features, and studied the accuracy and efficiency of identification methods, they cannot extend new identification method under existing architecture, and cannot identify unknown traffic.

The remainder of this paper is organized as follows: Section 2 proposes the architecture of EPFIA and dis-

cusses the impact of identification methods running sequence on system performance. Section 3 presents an online identification methods updating mechanism. Section 4 studies the method of using policy mechanism to maintain and update identification system. Section 5 validates the feasibility of EPFIA and tests its performance. Finally, Section 6 summarizes the whole paper.

2. Design of EPFIA

2.1. Design Principle

The main work process of EPFIA is as follows. Firstly, the packet capturer module is deployed at the bottom of EPFIA, which captures packets from the physical link, and filters packets that do not use TCP/UDP (Since P2P application usually using TCP/UDP transport protocol), and then submits packets to system identification modules in uniform format and style. In the paper, we use identification module denote the concrete implement of identification method. Using the uniform format so that it has packet information needed by any identification module, and using the uniform style because some identification methods only need one packet in a flow while others need a sequence of successive packets. Besides, the uniform format also makes all identification modules can work with the same packet capturer module. Since capturing packets from physical link consumes a large amount of system resources, it is of great important to use a uniform packet capturer module.

EPFIA handles packet based on flow state, which can be classified into two kinds: known and unknown. Known means a flow belongs to a kind of P2P or non-P2P application that has been identified, and it is of no need to identify it any more. While unknown means we need the identification module to handle the packet. Suppose Λ is the packet information, and its format is shown in **Figure 1**.

The flow identifier Λ_{flowID} is calculated using function CreateHash based on the first 5 field in **Figure 1**, which can be used to determine which flow the packet belongs to, and then determine whether the packet belongs to unknown.

Finally, the unknown packet should be handled by many P2P identification modules, and change the flow state to be known. During this process, we can use $\boldsymbol{f}_{\text{port}}$ to identify non-P2P flow based on Λ_{srcPort} and Λ_{dstPort}, and use $\boldsymbol{f}_{\text{sig}}$ to identify P2P flow that has application layer signature. For other unknown flows, we should design corresponding identification method to identify them.

source IP address ***srcIP***	source port ***srcPort***	destination IP address ***dstIP***	destination port ***dstPort***
transport protocol ***proto***	packet size ***size***	arrive time ***time***	flow identifier ***flowID***
application layer payload: ***payload***			

Figure 1. Format of packet information.

To achieve the goal of identifying many P2P applications in a system, we need to run many P2P identification modules simultaneously, update these modules frequently, and manage these modules remotely. As an extensible architecture for identifying P2P applications, we should focus on the following questions when designing EPFIA:

1. Whether these P2P flow identification modules should be run randomly or in some fixed sequence? If there are some relationships between efficiency and identification sequence, what are the relationships?
2. Since new identification methods keep on emerging, can we extend some new identification methods conveniently without recompile the existing program?
3. If we want to remotely update, start or stop a P2P identification module running at somewhere in the network, is there a good implementation mechanism?

2.2. Best Running Sequence of Identification Modules

EPFIA should call many specific identification modules so as to implement the identification of unknown flow. By changing the running sequence of many different identification modules, we can improve the whole working efficiency.

Suppose p(x) represents the computation cost of identification module x when identifying a flow, and $\boldsymbol{f}_{\text{port}}$ cost is $p(x_1)$, $\boldsymbol{f}_{\text{sig}}$ cost is $p(x_2)$, $\boldsymbol{f}_{\text{behavior}}$ cost is $p(x_3)$, $\boldsymbol{f}_{ML}$ cost is $p(x_4)$. Based on the identification principle we know the following inequation hold on:

$$p(x_1) < p(x_2) < p(x_3) < p(x_4) \quad (1)$$

During time T, if there are F flows, and the percentage of flows that identified by identification module x is $\mathbf{f}_x$, we give the following definitions:

Definition 1: Identification Module Running Sequence (**Identification Sequence** in short). **Identification Sequence** represents the set of identification modules and their running sequence. If we use R_n to represent the **Identification Sequence** of n different modules, then $R_n=\{r_1\rightarrow\ldots\rightarrow r_i\rightarrow\ldots\rightarrow r_n\}$, in which r_i represents the ith running modules.

Definition 2: System Identification Cost (**Cost** in short): The cost of identification sequence R_n when identifying F flows in time T. We use $\text{POWER}(R_n)_T$ represent **Cost**.

Definition 3: Priority Relationship P of Identification Modules (**Priority** for short). For any two identification modules in R_n, P represents the running sequence that makes the **Cost** smaller. Suppose the **Identification Sequence**

$$R_n=\{r_1\rightarrow\ldots\rightarrow x\rightarrow\ldots\rightarrow y\rightarrow\ldots\rightarrow r_n\},$$
$$R_n'=\{r_1\rightarrow\ldots\rightarrow y\rightarrow\ldots\rightarrow x\rightarrow\ldots\rightarrow r_n\},$$

if and only if $POWER(R_n')_T < POWER(R_n)_T$, we say y and x have **Priority P**, and note it as y**P**x.

Definition 4: Running Efficiency of Identification Module x (**Running Efficiency** for short). The ratio between the number of flows identified by x and the computation cost of x in time T is defined as **Running Efficiency** of x, which is represented as $E(x) = \mathbf{f_x}*F/p(x)$.

Suppose F flows belong to m applications, and there are n identification modules that can identify each flow accurately. These modules have **Priority P** and we can adjust their running sequence randomly, then we have the following lemma.

Lemma 1: There is a best **Identification Sequence** R_n that makes the $POWER(R_n)_T$ minimum.

Proof: For the n identification modules, there is a sequence R_n' that $\forall x \forall y(x, y \in R_n' \rightarrow xPy)$. If we exchange any two identification modules in R_n' and form R_n'', from definition 3 we know $POWER(R_n'')_T > POWER(R_n')_T$. That is, there is a best **Identification Sequence** R_n, in which any two identification modules have **Priority P**, so that $POWER(R_n)_T$ is minimized.

Theorem 1: Identification modules that have high **Running Efficiency** have high **Priority**, that is $\forall x \forall y(x, y \in R_n \wedge E(x) > E(y) \rightarrow xPy)$.

Proof: Suppose E(x) < E(y), x and y have the **Priority P**. Since the **Running Efficiency** of y is higher than that of x, by running y first we can reduce the number of unknown flows, and thus reduce the system identification cost, which conflicts with the assumption that the **Priority** of x is higher than that of y. Thus, when running more than one identification module, we should adjust their running sequence according to their **Running Efficiency** so as to reduce **Cost**.

2.3. Architecture of EPFIA

Suppose $p(x_2) = m_2p(x_1)$, $p(x_3) = m_3p(x_1)$, $p(x_4) = m_4p(x_1)$, then

$$1 < m_2 < m_3 < m_4 \quad (2)$$

Non-P2P traffic accounts for about 30% of the total Internet traffic, and most of them use fixed ports to communicate with each other [1], thus we can use port number to identify them. The traffic of BitTorrent and eDonkey takes about 80% of total P2P traffic, and more than 73% and 83% of them do not encrypt their payload, thus we can use application layer signature to identify them. Other types of P2P application flow have no port and application layer payload feature, and thus we should design specific identification methods for them. Take the former discussion into consideration, the efficiency of $\boldsymbol{f}_{\mathbf{port}}$ is $E(x_1) \approx 0.3*F/ p(x_1)$, and the efficiency of $\boldsymbol{f}_{\mathbf{sig}}$ is $E(x_2) \approx 0.41*F/p(x_2)=\frac{0.41\times F}{m_2 \times p(x_1)}$, and the efficiency of $\boldsymbol{f}_{\mathbf{behavior}} + \boldsymbol{f}_{\boldsymbol{ML}}$ is $E(x_3 + x_4)\approx \frac{0.29\times F}{p(x_3)+p(x_4)} > \frac{0.29\times F}{m_4 \times p(x_1)}$.

Each identification module identifies the application of flow according to $\boldsymbol{\Lambda}$, and we can use the number of packet to estimate the computation cost of a special identification module. Based on the principle of different identification methods, we know $m_4 \geq m_3 \geq 3$, and thus

$$E(x_1) > E(x_2) > E(x_3 + x_4) \quad (3)$$

As a result, we can get the best efficiency by using the following **Identification Sequence**: firstly using $\boldsymbol{f}_{\mathbf{port}}$ to identify non-P2P flows, and then using $\boldsymbol{f}_{\mathbf{sig}}$ to identify un-encrypted P2P flows, finally using other special methods to identify the remainder P2P flows.

Observed that the identification is based on flow, and as the process of identification, the percentage of identified flows will keep on increase. If there are M packets in a new flow $\boldsymbol{f}_{\mathrm{new}}$, and an identification module can identify the application of $\boldsymbol{f}_{\mathrm{new}}$ based on the first n packets, thus the remainder M-n packets will need not to be identified any more. Generally M >> n, we can increase the identification efficiency by identifying which flow the M-n packets belong to. Thus we design a flow differentiation module, which identifies whether a coming packet belongs to a known or an unknown flow, and we only handle packets that belong to unknown flows. Experiment results show that this method can filter about 98% packets, and thus decreasing the identification cost effectively.

Based on the upper discussion, we designed an extensible P2P flow identification architecture, which is shown in **Figure 2**.

EPFIA uses flow identifier $\boldsymbol{\Lambda}_{\mathbf{flowID}}$ as the address of Hash-Table item, and this address is also used as the first address of storing active flow information ***flow-key***. ***flow-key*** = {source IP address srcIP, source port srcPort, destination IP address dstIP, destination port dstPort,

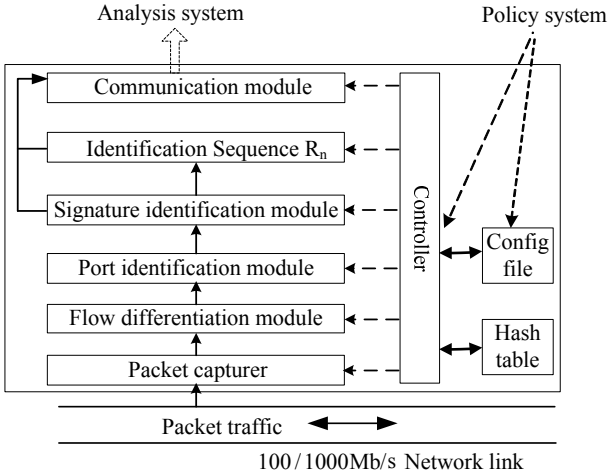

Figure 2. Extensible P2P flow identification architecture.

transport protocol Proto, flow state State }, in which State ∈{−1, 0, P2P_ID }, and "−1" represents unknown flow, "0" represents non-P2P flow, and "P2P_ID" represents the P2P application code defined by us. In addition, each flow has a counter, which is used to record the number of packets identified for unknown flows. When counter is larger than a threshold Φ, we change the state of that corresponding flow to be a non-P2P flow. We also use $[PORT]_{std}$ to represent standard network service ports, and use $[SIG]_{P2P}$ to represent the set of specific P2P application signatures.

3. Identification Program Online Updating Mechanism

In order to add new identification module, maintain and update current module and adjust the running sequence of different modules conveniently, EPFIA uses the form of plug-in to manage different identification modules. EPFIA stores all identification modules in a plug-in database, and stores the running sequence R_n in a policy database. **Figure 3** shows the format of plug-in stored in policy database.

In which "Priority" is a number that defines the plug-in's running sequence; and "P2P application code" denotes the P2P application that this plug-in can identify; and controller uses "Store path" and "Plug-in name" to locate the plug-in, and use "Main control function" to run this identification module. **Figure 4** shows the online updating mechanism of EPFIA.

During the initialization phase, the main control module accesses the policy database to get R_n, and then stores all plug-in information in the memory using list structure, which is sorted by priority. During the identification phase, the controller calls different identification modules to handle each packet according to the priority sequence.

EPFIA updates identification module by maintaining the plug-in database, and adds new identification module as well as adjusts the running sequence of different identification modules by handling the list, thus it avoids recompiling the program to implements such functions.

Plug-in name	*Store path*	*Main control function*	*P2P application code*	*Priority*

Figure 3. Format of plug-in stored in policy database.

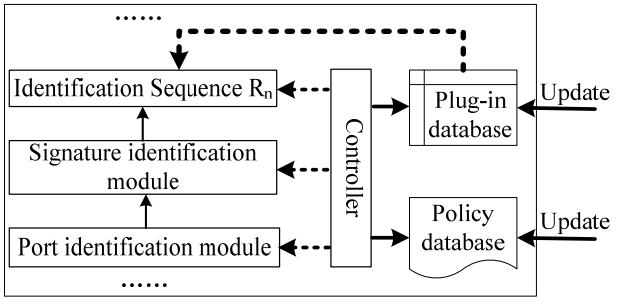

Figure 4. Online updating mechanism of EPFIA.

4. Policy Mechanism

EPFIA uses policy mechanism to support updating, starting and stopping identification modules remotely. The policy is divided into two kinds: manually and automatically. The manual policies are setup by administrator, which include identification modules running sequence R_n and parameters; automatic policies are generated using policy control language by EPFIA during the system running time, and they optimize the system efficiency by adjusting R_n dynamically. **Figure 5** shows the EPFIA policy mechanism.

In EPFIA, the controller provides two access interfaces: one is the plug-in database access interface FAP, and the other is the policy database access interface PAP. Administrators can access FAP and PAP through control center. The controller stores the received plug-in in the plug-in database, and stores the received policy in the policy database, and then updates R_n according to the new policy.

According to **Lemma 1** and **Theorem 1**, the running sequence of P2P flow identification modules can affect the efficiency of EPFIA. Though administrators can setup the running sequence of identification modules, it is very difficult to estimate the **Running Efficiency** of each identification module accurately and adjust their **Priority** immediately. EPFIA uses automatic policy and adjusts the module **Priority** dynamically. EPFIA determines the relative computation cost m_x of program x according to the number of packets x needs to handle. $F(x)_T$ represents the number of flows x identified during time T, and we use $E(x)_T = F(x)_T/m_x$ to represent the efficiency of x during time T. The automatic control policy computes the current identification efficiency $E(x)_{current}$ of x based on $E(x)_T$ and the old identification efficiency $E(x)_{old}$.

$$E(x)_{current} = (1 - \varepsilon)\cdot E(x)_T + \varepsilon\cdot E(x)_{old} \quad (4)$$

In Equation (4), ε is a decimal fraction between 0 and 1, which is used to control the sensitivity of efficiency

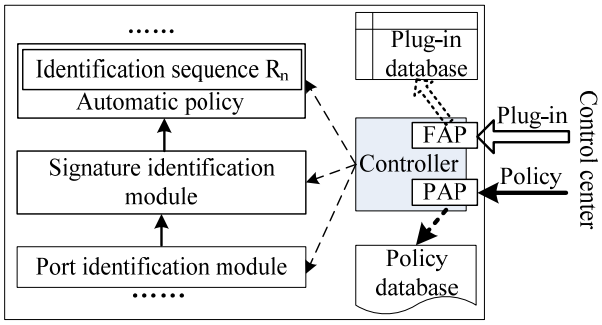

Figure 5. EPFIA policy mechanism.

variation. The automatic control policy calculates the current identification efficiency $E(x_i)_{current}$ of module $x_i(1 \le I \le n)$ periodically, and determines the **Priority** according to $E(x_i)_{current}$. Our experiment results show that the automatic control policy in EPFIA can adapt to the network traffic variation and increase the whole identification efficiency.

5. Experiments and Analysis

To validate the upper analysis, we developed the prototype system of EPFIA named P2P-Analyzer, which includes two P2P flow identification programs: one for UDP based Skype voice flow (Skype_Detection, SD), and the other for PPLive video flow (PPLive_Detection, PD). We test the performance of P2P-Analyzer for a long while in the campus network of PLAUST, and following is the typical experiment result and its analysis.

5.1. Experiment Environment and Method

The experiment environment of P2P-Analyzer is shown in **Figure 6**. The campus network connects to the CERNET through 100 Mbps fiber, and the experimental hosts have Intel Core2 CPU with 2.33 GHz frequency, and 2 GB memory. In the campus network we install Skype, PPLive and BitTorrent client, which run randomly during the experiment. The running information such as communication begin time, number of communications and download file size are also recorded manually. The Tcpdump is running at the mirror port of the switch so as to capture the packet information with 50 bytes application layer payload. The P2P-Analyzer is also run to identify P2P flow automatically, and send the identified P2P flow information ***flow-keys*** to P2P-Looker to handle. P2P-Looker is the software that can analyze multi-type P2P flow information, so we can decrease the influence of analyzing on P2P-Analyzer performance. Besides, we also setup a control center so as to manage P2P-Analyzer remotely.

In order to compare and analyze the performance of P2P-Analyzer, we firstly analyze the P2P flow information manually using data captured by Tcpdump, and use this result as the ground truth. Then compare it with the results identified by P2P-Analyzer. The counter is set in each module of P2P-Analyzer to record packets handled. For every 5 minutes the CPU usage of each module is calculated.

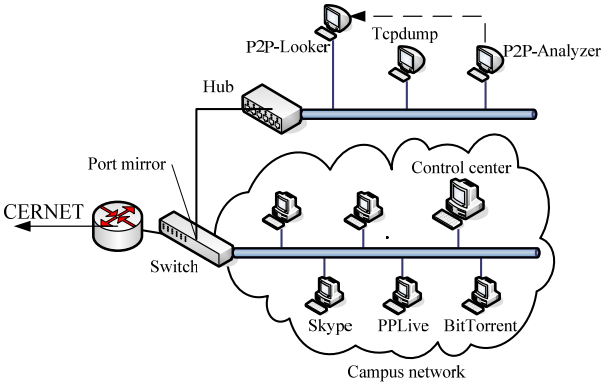

Figure 6. Experiment environment of P2P-Analyzer.

5.2. Experiment Results and Analysis

We analyze two typical experiment results carried out at different time. The packet information collected by Tcpdump is stored in two trace files, and the manual analysis result is shown in **Table 1**. Others include HTTP, FTP, DNS, Flash, FTP, ICMP, IMAP, MSN, POP, QQ, SMTP as well as unknown traffic, of which HTTP and Flash is more than 85%.

In the first experiment, SD runs in the first two hours, and PD is called by policy in the last two hours. We setup $\Phi = 10$ (the maximum number of packet that SD needs to identify Skype). The counter is outputted every hour so as to examine the performance of EPFIA. The result is shown in **Table 2**, from which we can see that the packet capturer filters 1% non TCP/UDP traffic, and the flow differentiation module filters 98% of the total packet, which decrease the load of identification modules dramatically. The number of unknown packet is decreased after calling PD. In addition, PD can identify PPLive flow before the number of packet reaches Φ, and thus decreases the load of other identification modules further.

Figure 7 shows the CPU usage during the first experiment. In the first two hours when running SD, the average CPU usage is 11.3%, and in the last two hours when running SD and PD together, the average CPU usage is 10.1%. From this we see that when adding new module, the usage of CPU can decrease because of the **Running Efficiency** of identification modules.

In the second experiment, we setup the relative computation load of PD and SD according to the number of

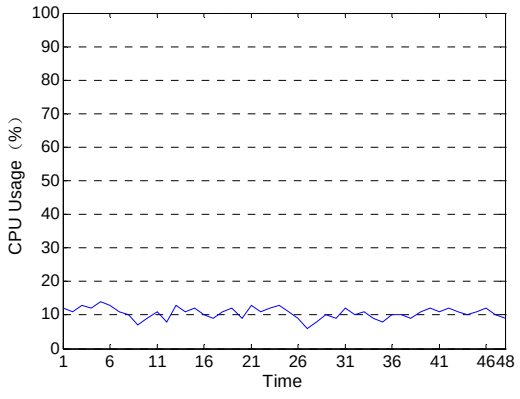

Figure 7. CPU usage information during the first experiment.

Table 1. Manual analysis result of two traces.

Traces	Start	Dur	Flows					Packets				
			Total	*Skype*	*BT*	*PPLive*	*Others*	*Total*	*Skype*	*BT*	*PPLive*	*Others*
T1	Mar20 08:00	4 h	99.3K	0.3K	51K	30K	18K	89M	6.2M	48.6M	20.2M	13.9M
T2	Mar21 14:00	5 h	129.7K	0.7K	62K	44K	23K	113.6M	13.7M	51.7M	29.8M	18.3M

Table 2. Number of packets disposed by different modules in P2P-analyzer.

Module name	First two hours		Last two hours	
	packet number	*percentage*	*packet number*	*percentage*
Packet capturer	43 M	-	46 M	-
Flow differentiation module	42.6 M	99%	45.6 M	99.1%
Port identification module	379 K	0.86%	215 K	0.46%
Signature identification module	371 K	0.84%	207 K	0.44%
PPLive_Detection(PD)	-	-	185 K	0.39%
Skype_Detection(SD)	351 K	0.8%	170 K	0.36%
Unknown packets	350 K	0.79%	169 K	0.35%

packets needed to identify a flow. That is $m_{PD} = 4$, $m_{SD} = 10$. Using Equation (4) to calculate the identification efficiency of each module, and set $\varepsilon = 0.5$, $E(PD)_{old} = E(SD)_{old} = 0$. We use policy mechanism to call SD and PD one after the other, so as to test the impact of automatic policy on system efficiency.

By analyzing the traffic data further, we see that there are 174 Skype flows in the first hour, and 11264 PPLive flows. According to automatic policy mechanism, in the second hour, the sequence of SD and PD should be adjusted. **Figure 8** shows the number of packets SD and PD handled per hour, which includes Skype, PPLive and other unknown application packets. From **Figure 8** we can see that in the first hour the number of packets handled by SD is a little larger than that of PD. This phenomenon demonstrates that by adjusting the running sequence of SD and PD according to their identification efficiency, the number of packets handled by SD decreased dramatically, and the identification efficiency of P2P-Analyzer increased.

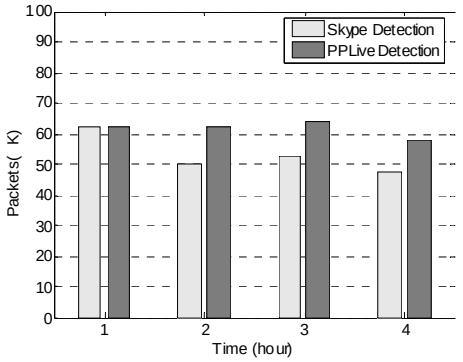

Figure 8. Number of packets SD and PD handled per hour.

6. Conclusion

Taking the problems of identifying large number of P2P flows into consideration, in this paper we propose an extensible P2P flows identification architecture (EPFIA). In order to obtain the highest efficiency of EPFIA, many identification methods should be arranged following the optimizing sequence. An online mechanism of renewing identification methods is designed, which can extend new P2P identification method without recompiling the whole program. Applying policy mechanism, identification modules can be updated, started and halted with policy remotely. The experiment results of running the prototype system show us that EPFIA could effectively promote the performance of system and support online renew P2P identification methods and manage them remotely. In the further works, we will focus on deploying the system in large scale, and develop application systems based on P2P traffic analysis.

REFERENCES

[1] H. Schulze and K. Mochalski, "Internet Study 2008/2009," Technical Report, Ipoque GmbH, 2009.

[2] K. C. Claffy, "Internet Traffic Characterization," University of California, San Diego, 1994.

[3] T. Karagiannis and A. Broido, "Is P2P Dying or Just

Hiding?" *IEEE GLOBECOM* 2004, Dallas, Texas, 29 November-3 December 2004, pp. 1532-1538.

[4] S. Sen and O. Spatscheck, "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures," *WWW*2005, New York, USA, 17-22 May 2004, pp. 512-521.

[5] M. Roughan and S. Sen, "Class-of-Service Mapping for QoS: A Statistical Signature-Based Approach to IP Traffic Classification," *IMC* 2004, Taormina, Italy, 25-27 October 2004, pp. 135-148.

[6] A. Moore and K. Papagiannaki, "Toward the Accurate Identification of Network Applications," *PAM* 2005, Boston, USA, 31 March-1 April 2005.

[7] T. Karagiannis and A. Broido, "Transport Layer Identification of P2P Traffic," *In IMC*'04, Taormina, Italy, 25-27 October 2004, 14p.

[8] T. Karagiannis and K. Papagiannaki, "BLINK: Multilevel Traffic Classification in the Dark," *SIGCOMM*'05, Philadelphia, USA, 21-26 August 2005, 12p.

[9] L. Bernaille and R. Teixeira, "Early Application Identification," *The* 2*nd ADETTI*/*ISCTE CoNEXT Conference*, Lisboa, Portugal, December 2006.

[10] M. Crotti and M. Dusi, "Traffic Classification through Simple Statistical Fingerprinting," *SIGCOMM Computer Communication Review*, Vol. 37, No. 1, 2007, pp. 5-16. http://dx.doi.org/10.1145/1198255.1198257

