Application Behavior Analysis by Stateful Automata Mechanism

Nen-Fu Huang[1,2,*] Yi-Hsuan Feng[1]

[1] Department of Computer Science

National Tsing Hua University

Hsin-Chu 300, Taiwan, ROC

[2]Institute of Communication Engineering

National Tsing Hua University

Hsin-Chu 300, Taiwan, ROC

{nfhuang, dr918302}@cs.nthu.edu.tw

Received 16 November 2007; Revised 30 November 2007; Accepted 9 December 2007

Abstract. A sufficient visibility into the behaviors of network applications from the Internet traffic is essential to the content security, traffic management, and measurement. This paper presents a methodology to perform a reliable traffic classification and distinguish activities of specific applications. Our approach uses the flow-based state machine to model a given network application and its behaviors (even with the encryption) and combines the signature matching, protocol analysis, and statistical test in order to make use of the strength of the three approaches. We further discuss the system design and the implementation of our framework, including the detection heuristics and system details. These systems are already deployed at the borders of network environments of several enterprises and organizations. At last, we demonstrate the effectiveness of the approach by applying it to identify various applications and malicious traffic. This study on application behaviors shows that it is possible to allow the expected activities of programs but disallow others between the endpoint users.

Keywords: application classification, stateful method

1 Introduction

The network administrators usually impose a set of rules carefully on their networks to enforce the security policy (e.g., no delivering of files by FTP, P2P, or mail attachment) and protect the network resources from unnecessary consumption (e.g., the limited bandwidth budget exhausted by gaming). Two prerequisite requirements in this task are:

- An accurate identification on applications and behaviors, and the continued tracking of them
- Different reactions for different behaviors.

Since the modern routers and firewalls have the ability to look for the features in the TCP and UDP headers, the port-based methods [1, 2] relying on the well-known service ports are the simplest approach for identification of a particular application. However, the emerging problem comes from new applications either not using the native port numbers or using other protocols, such as HTTP, as tunneling to go through the firewall without being blocked. It is well known that the identification by such a port-based heuristic is found to be no longer accurate. An example is the underestimation up to 70% of the popular Kazaa P2P traffic in [3].

The observations on port-based identification emphasize the imperativeness of a deeper inspection into payload and into protocol-specific semantics to offer a more reliable detection. First, several works [3, 4, 5, 6] use the application information in the payload content to define protocol-specific signatures and then check whether a flow carries these byte-string patterns in payloads. In more details, a signature is a "fingerprint" describes uniquely a set of features (or patterns) of an input data for the packet inspection. Each of these patterns has its type (e.g., hex value and string) and value domain. The signature-based method is a common technique used in IDS, such as Snort [7], and other industry security products. At the other end, protocol analyzers [8] check whether a traffic stream follows application-level semantics to determine the type of traffic. With the specific knowledge of an application, a protocol analyzer uses a set of detection heuristics for the data it receives and plays as a dissector to extract the information when necessary. For example, the HTTP analyzer helps to ensure

* Correspondence author

the traffic through port 80 adheres the HTTP specification for detecting the tunneling of instant messaging and P2P software.

By using the statistical analysis of traffic, the previous works [9, 10, 11] give a different point of view on the traffic classification. These techniques use different parameters including the packet size distribution and the interactive relationship to classify the traffic into board categories.

For identifying the application behaviors, the existing methods have the some limitations. First, each application has two phases that need to be tracked: the connections associated with the application and the packets with important information. However, previous works have focused on the detection performance (evaluated by the false positives and false negatives) of connections and did not discuss the requirements of behavior detection. For instance, an enterprise network administrator might constitute a policy to block the file exchange by IM software for the security and bandwidth management but allow the message exchange between the company's branches to save the cost of phone calls. Therefore, to apply the different enforcement policies to different behaviors, a deeper visibility into the specific application is necessary. Second, the application detection by port-based, signature-based, or statistical approach only provides a coarse-grained information of whether a given protocol is in use or not. However, this black-or-white result is not enough and it is important for the behavior detection to have the capability of analyzing every application instance continuously and analyzing the parent connections to identify their children connections. For example, many protocols (e.g., H.323) establish connections and negotiate service parameters on well-known TCP ports and then establish another ephemeral connection to transfer the following data. Thus, we need to track the control connections on "well-known" ports that spawn "ephemeral" data connections on arbitrary ports.

For the purpose of recognizing the different behaviors, we combine the techniques of the signature matching, protocol analysis, and statistical test for making use of their strength in order to provide a complete framework. We demonstrate the effectiveness of the approach by four popular examples: eDonkey, BitTorrent, MSN, and Yahoo Messenger.

The remainder of the paper is organized as follows. In Section 2, based on our previous study, we have defined the application classes and behavior classes. Next, Section 3 presents our approach and the formal definitions, and also gives an FTP example to explain our framework. In Section 4, we discuss the system design and implementation, including the detection heuristics and system details. Section 5 introduces the specific automata of four popular IM and P2P protocols. Section 6 evaluates the detection performance of our approach and comparing it with other identification methods. We also give a report of the hottest application activities monitored at a gateway node. Section 7 gives a conclusion and future works.

Table 1. The application classes and behavior classes identified in this study

Application class	Behavior class	Example protocol/application
Web browsing	http get, http post, file download, cookie, Java applet/ActiveX	WWW
Instant messaging	login, chat, file transfer, audio/video communication, web IM, on-line game	MSN, Yahoo Messenger, Google Talk, ICQ, AIM, Skype
P2P	login, file query, peer list, file sharing	Gnutella, BitTorrent, FastTrack, eDonkey2000, WinNy
Bulk	login, file download, file upload, list directory content, active mode, passive mode	FTP, GetRight, FlashGet
Email	authentication, file attachment	POP3, SMTP, IMAP4
Others		Malicious traffic (e.g., DNS flood)

2 Identifying Application Behaviors

To identify the application behaviors, we studied a suite of thirty applications. Based on this study, we identified a set of *application classes*, their *protocols*, and their *behavior classes*. For each application class, we identified the behavior classes commonly used to implement the primary functionalities in the class of application. However, some applications do not really implement these primary functionalities. For example, for instant messaging (IM), Microsoft MSN Messenger, Yahoo Messenger, AIM, and ICQ all support the behavior classes of login, chat, file transfer, and audio/video communication, but other IM software (e.g., Google talk) only supports partial functionalities. Note that we consider these behavior classes as simply the *unions* in the corresponding application classes, and not as the *requirements*.

Table 1 presents the application classes, behavior classes, and protocols that we identified. We classify the protocols into their classes based on the user experience with the applications and user interfaces. For example, Skype [14] allows users to send messages, make a conference call, and transfer files to other users in the net-

working. Though Skype relies on the P2P technology, we still classify it into the category of instant messaging. We do not claim that the structure of classification presented in Table 1 is either unique or comprehensive, and based on further experience and study, we expect this classification table can be extended and refined.

3 Proposed Approach

A given network application and its behaviors can be modeled using *Flow-based Automaton* (FA), which augments the traditional finite state machine with statistical properties. A transition is a state change triggered by a particular input event, i.e., transitions map some state-event pairs to other states. A transition condition is evaluated by the appropriate analysis method. In our design, the analysis methods include the techniques of signature matching, protocol analyzer, and statistical test. An event-driven FA M starts from the initial state s_0 and reads the event as its input symbol. Whenever the machine M accepts the event, it changes to the next state and executes the action component associated with the transition. In general, the FAs are non-deterministic.

We augment the flexibility of state machines with the statistical properties that can be used to detect the particular traffic. Our attribute set includes only the easily computed operations and are all bidirectional meaning. The potential events are first identified by an analysis method (e.g., signature matching) and are verified by the statistical parameters of the transition in terms of:

- The *repetition frequency* which a particular transition condition in the FA is fulfilled per given time period
- The *inter-arrival time* between two events
- The *current flow statistics* which are continuously updated counters, e.g., the total packet number, total byte number and flow up-time.

In summary, each FA delineates a sequence of transitions between the packets of application flows to model the behaviors. The transition condition can be verified by the signature-based method, protocol analysis, and statistical properties. Rather than just the content of a single packet, we track relationships between packets, flows, and applications.

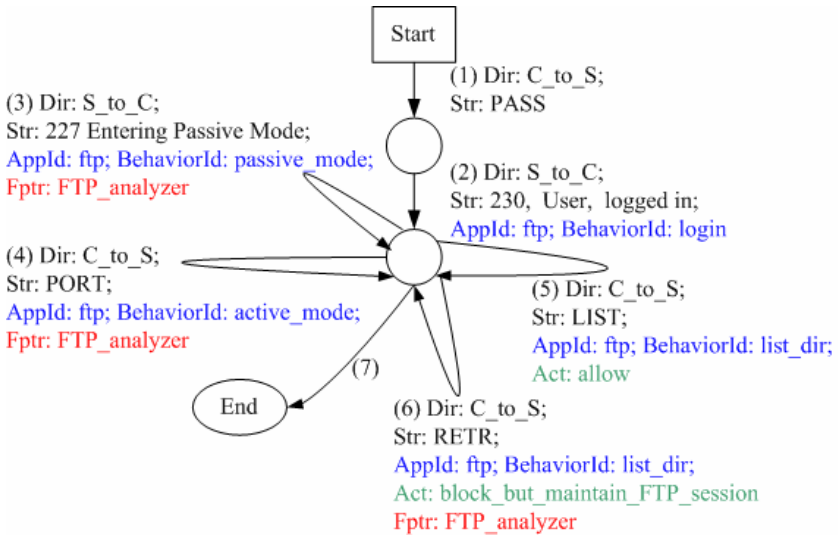

Fig. 1. The FA of FTP protocol

To understand how such FA specifications can be used for monitoring protocol and behavior, consider the FTP FA in Fig. 1. We assume that a gateway router (with FTP FA) connects an internal network of organization to the Internet and bi-directional traffic is under the surveillance.

Fig. 1 shows that the FTP FA relies on signatures to model the application behaviors and uses FTP protocol analyzer to proactively detect the dynamic port used for the children data connections both in active mode and in passive mode. First, note that we do not select the well-known port (21/tcp) of FTP service as features in order to avoid the misclassification due to port detection, and rely on the direction attribute (e.g., from the client to the server) and byte-string patterns. In addition, we classify a flow as the class of FTP when the FA accepting *transition 2*. The 4-byte-only string pattern of *transition 1* raises rapidly the possibility of false positives. However, two effective transitions in different packets of a flow give us the confidence of accuracy. It shows that we have more flexibility in how we carry out the detection. Second, we delegate the FTP protocol analyzer activated by the

function pointers in *transition 3* and *4* to extract the information of dynamic port of oncoming data connection, and to register this data connection by a 4-tuple (i.e., source IP, source port, destination IP, destination port) to a flow database. Thus, we can classify immediately the oncoming data sessions as FTP class when the first packet received. Furthermore, for the security management, any FTP data connection without the registration in the flow database can be blocked. Third, FTP FA enables us to assign different enforcement rules to different behaviors. The *transition 5* and *6* represent the behaviors of listing the directory content and downloading a file respectively. This FTP FA allows the `LIST` command but modify the `RETR` command to "`R_T_`" on-the-fly to disallow the download attempt, and this also makes a FTP server ignore this request only and still maintain the FTP control session seamlessly. Finally, the *transition 7* closes a FTP instance because of entry timeout or TCP FIN/RST packet from the control session.

We conclude this section with the benefits of our approach as follows. Our approach:

- *provides a higher accuracy from the flow's point of view*. Our approach provides a framework to model active behaviors by a sequence of explicit state transitions from packets in application flows. With the strength of statistical test, a higher accuracy comes from the looking for the exact match in the byte strings in a flow, i.e., not only in a single packet, by the signature matching and protocol analysis.
- *simplifies the process of feature selection from the packet's point of view.* The quality of signature-based traffic classification depends on the quality of signatures, especially on the discriminating byte-string patterns. If these patterns are not selected carefully (e.g, long enough or explicit enough), the possibility of false positives or false negatives rises. Thus, the signatures need to be crafted as tight as possible. However, as shown in the above FTP example, in our approach the multi-state analysis allows us to use multiple byte-strings as patterns for individual signatures which are located in the different packets, and makes the pattern selection easier while not decreasing the detection efficacy.

3.1 Flow-based Automaton Definition

Below, we describe the definitions for specifying FA that models the application classes and behavior classes.

Definition 1: An FA is denoted by a 3-tuple (S, Σ, δ), where

- S is a finite set of application states such that $S = \{s_0, s_1, s_2\ldots, s_{n-1}, s_f\}$. The s_0 is the start/initial state, and the s_f is the final state,
- Σ is the alphabet of events or input symbols,
- δ is the transition function that maps $S \times \Sigma$ to S.

Definition 2: An input event e is represented by a 2-tuple (ID_{app}, $ID_{behavior}$), where

- ID_{app} is the identity of application/protocol,
- $ID_{behavior}$ is the identity of behavior class in a particular application class.

Definition 3: We denote a transition δ by a 3-tuple (Q, A_{eff}, F_{eff}), where

- Q is the set of transition conditions needed to be satisfied, such that $Q = \{q_0, q_1, q_2\ldots, q_{i-1}\}$,
- A_{eff} is the transition action,
- F_{eff} is the function pointer used for invocating the external function (i.e., protocol analyzer).

Here, the analysis methods should confirm the fulfillment of the set Q. An event e occurs when a set Q is satisfied and it may indicate the application and behavior class (if the identities exist) of a flow. Note that an event is not necessarily identical to a mark of application or behavior identification, and it can be used for the intermediate state in the transition path of an FA in order to enhance the accuracy of classification. For instance, the *transition 1* in above FTP FA is not used to classify a flow, and only to enter the next intermediate state. The component A_{eff} consists of the actions will be taken for an effective transition. Allowable actions include the next-state assignment in FA, the alarm notification, the function invocation by F_{eff}, and the interference to the packet, and to the flow/application which it belongs to.

4 System Design and Implementation

In this section, we introduce the structural block diagram of our approach and the detection heuristics in our implementation. Fig. 2 presents the functional structure of the FA online classification. We assume the classifier runs either at the border router or in the monitoring host that is able to process all packets of bidirectional traffic in the link.

A *flow tracking* module extracts the 5-tuple (i.e., transport protocol, source IP, source port, destination IP, destination port) of a received packet and gets a data structure of the flow this packet belongs to. This per-connection data structure (called `flowBuf`) contains the information needed for the remainder of classification

processes, for example, the statistical attributes, the application class, behavior class, and the snapshot of the interaction between the client and server. The *policy* is a FA collection which follows the language defined in Section 3.1. A policy contains the finite states, transition conditions, events and the mapping table among previous elements. After the analysis of *Signature Matching* and *Statistical Test* module, the *Transition Mapping* module is responsible to update the information of active FA instances and generate an event to the next *App&Behavior-class Identification* module. In details, for each transition from the initial state of FA, the *Transition Mapping* module adds this active instance to the corresponding `flowBuf` structure and deletes it when a state machine reaches to the final state.

The *Flow Registration* module updates the application class and behavior class of a `flowBuf` or generates a new `flowBuf` entry for an oncoming connection. The *post-detection engine* executes the action component of transition, assigns the system path (path 1 to 3 in Fig. 2) for the following packets of a flow, and also activates the protocol analyzer if necessary. For example, an email analyzer of SMTP/POP3/IMAP protocols is used to strip the MIME-type encoding to get the attachment information and to monitor the authentication process of a mail exchange. Before the packet transmitted to the network link, the *Flow Statistics* module updates the statistical flow attributes in `flowBuf`.

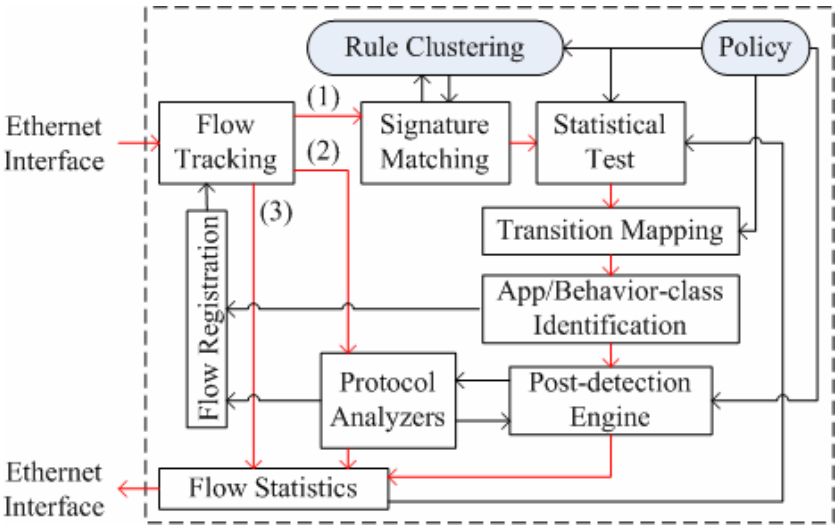

Fig. 2. Design of online application and behavior identification

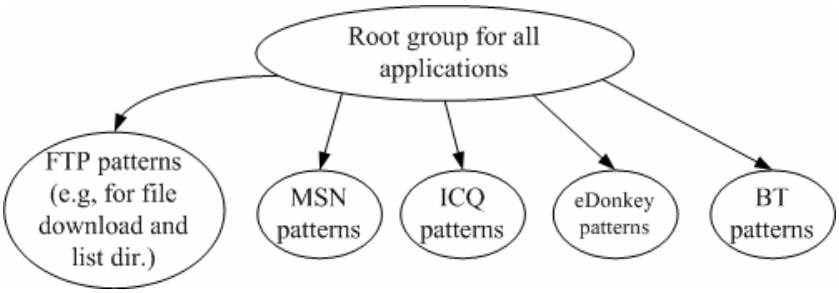

Fig. 3. The example of rule clustering

A question of application classification is *when* to stop the inspection into a classified connection. Though the previous works [3, 8, 12] indicate that it is sufficient for protocol detection by the few examination on the first KBs at the beginning of a connection, we found that sometimes it is necessary to monitor an entire flow stream for classifying behaviors. For instance, since an authenticated MSN client connects to a so-called SwitchBoard server by a single TCP connection and all chat messages and file transfers pass through this connection, we have to monitor a MSN connection to differentiate the behaviors until its close. Different from the assumption in [8], when monitoring protocol behavior, we allow an active flow to be adjusted for its application class, protocol and behavior class dynamically. For example, consider a user who uses the GetRight utility to download a file while browsing a web page. At first, this flow is classified as WWW because of passing a URI of the file to the server, but then classified as the file download behavior of GetRight software by the following packets. On the other hand, if many classified flows can be forwarded immediately to the outgoing interface by the path 3 in Fig. 2 for

bypassing the remainder of classification processes, then overall classification performance can be improved. Thus, this question should be answered on a per-protocol basis. In summary, long term data-intensive flows, including file up/download sessions of Bulk and P2P classes, follows the path 3 in Fig. 2, flows of SMTP/POP3/IMAP protocols are forwarded to the path 2 and the rest enters the path 1.

The technique *Rule Clustering* on signature matching especially is used to yield a better runtime performance by minimizing the redundant comparisons. In the pre-classification phase, the transition conditions belonging to the signature matching are partitioned into smaller subsets, and the mechanism used to select rule subsets bases on the transition level and protocol. First, all transitions originated from the initial states are collected into a root group, namely all *transition 1s* of FAs. Next, the other signatures are grouped according to the applications. Fig. 3 shows the logical view of a rule clustering example. When the application of a flow is already determined, we only perform the necessary signature matching in the root group (remember that a flow can be determined into a different application dynamically) and the group for current behavior classes.

Apart from the SMTP, POP3, and IMAP protocols, we use the signature matching as the primary heuristic to detect the potential applications and their behaviors. The signature patterns and the design of protocol analyzers are derived from RFCs, the description of applications, and the network traces. To our knowledge, before the major upgrade of software implementations, these protocols change nothing or just a little. This characteristic makes the effort of FA modification low, and main challenge comes from the preliminary FA design. As described before, the quality of patterns is important for building efficient FAs and not only the port range of transport layer and strings in payload can be the discriminating features. Sometimes, a FA with a sequence of packet sizes is very useful. For example, we observe that some behaviors of Skype and WinNy even with encrypted communication have their unique models regarding the packet sizes and can be identified by this feature without false positives. However, to identify the behaviors within these encrypted flows by string matching is impractical or impossible. In addition, the per-flow average packet size is also used to refine the classification [9].

A transition can activate a protocol analyzer to decode a payload/TCP stream (e.g., email analyzer), to extract the protocol information (e.g., ftp analyzer or BitTorrent analyzer), or to normalize a URL (http analyzer). In addition, the experiment results of [12] show that the SMTP/POP3/IMAP protocols are detected exclusively by their official ports, because an email must be exchanged with other sites and other hosts and the random nature of the ports is useless for mail services. Thus, flows on ports 25/tcp, 110/tcp and 143/tcp are sent to the email analyzer directly by the path 2 in Fig. 2.

We implement our approach on an embedded system powered by an ARM9200/200MHz SoC, which comprises a string searching engine and the system memory is 128MB SDRAM. The operating system is Linux kernel 2.4.29. The SoC contains a string searching engine by a tree-based Aho-Corasick algorithm. Total system runtime memory size is less than 32MBs. To get the best performance, we do not utilize the `netfilter` framework and develop three *tasklets* to receive the packets from network driver, perform the functionalities in Fig. 2, and transmit the packets to driver directly. This system is already deployed as border routers (total over 20 sites) of organizations to comprehensively evaluate the classification accuracy (i.e., the quality of signatures and protocol analyzers) and the mechanism of behavior control. Our implementation and policy (total over 30 applications) have been examined by over 1 year with numerous discussions and mail exchanges between these network administrators.

5 Examples of Application-specific Automata

This section gives other FA examples of stateful applications, including MSN, Yahoo Messenger, eDonkey2000, BitTorrent and a flooding detection. By following the pre-defined transactions, the communication and execution path of these applications are predictable. We also use the protocol analyzers for different applications to extract the peer list or execute the security management.

5.1 P2P: eDonkey and BitTorrent (BT)

eDonkey is a server-based file sharing protocol. The standard listening port of eDonkey file sharing is 4662 for TCP, but a user can modify it manually. The data chunks transferred between the peers are compressed, and the discriminating string pattern is very short in individual packets. Again, we use multiple transitions (the path from *transition 1* to *transition 3*) to model its data sharing to maintain the accuracy. The bottom automaton in Fig. 4 shows the characteristics that eDonkey peers share a file by delivering data chunks repeatedly in *transition 5*. The top automaton in Fig. 4 presents the login process initiated from the client. In addition, we found that an eDonkey server sends a simple *ping-pong* test for reachability even to a host which has no eDonkey software installed, and it is easily considered as a misclassification.

We depict the BitTorrent FAs in Fig. 5 and they are much concise compared to eDonkey's because of the simplicity of BT protocol. Through the meta-info (torrent), a BT tracker will reply a peer list containing the listening ports to the downloader. By the operations of decoding and extraction in BT analyzer, we register these flows activated by peer list to the *Flow Tracking* module for the download phase.

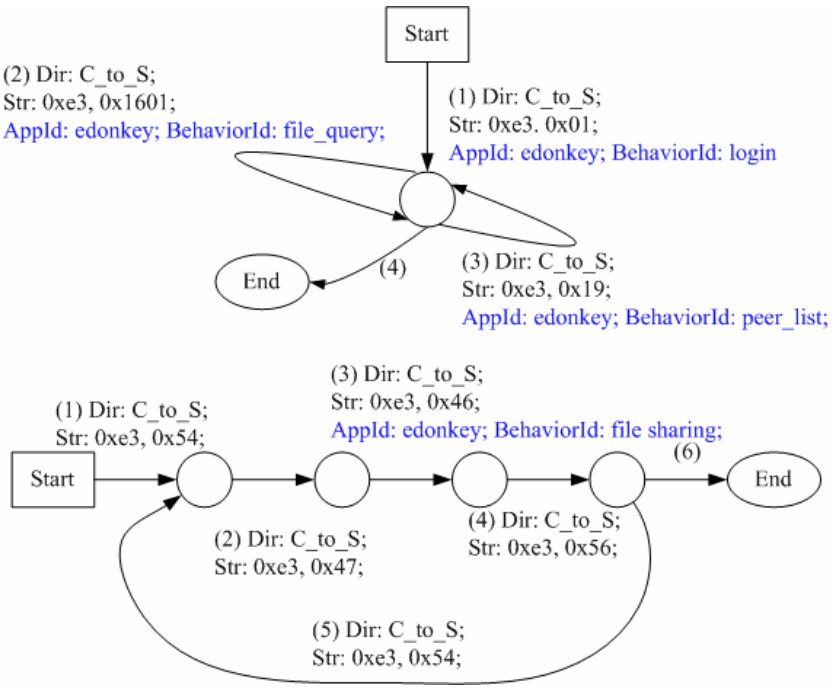

Fig. 4. Bidirectional FAs for the authentication and data sharing phases of eDonkey

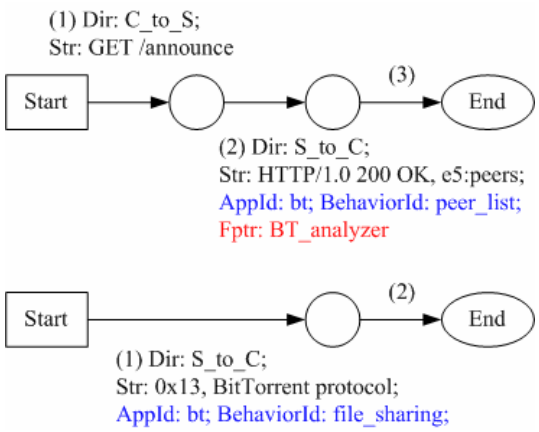

Fig. 5. The BitTorrent FAs for capturing the peer list

5.2 IM: MSN and Yahoo Messenger (YMSG)

MSN and YMSG are very popular systems of the presence and instant messaging, and RFC 2778 [15] provides a general overview of what these systems do. Here, we only illustrate the behavior classes of login, chat and file transfer, and omit the audio/video communication, and on-line game. Besides the behavior identification, we also provide the keyword search in the instant messaging of MSN and YMSG services.

Besides the official communication over TCP port 1863, a MSN client can uses the HTTP tunneling and SOCKS proxy to bypass the firewall. Fig. 6 illustrates the two FAs over TCP port 1863 (the bottom) and two FAs over HTTP (the top). A MSN client has to authenticate a Notification Server first, and then connect to a Switch Board Server for instant messaging.

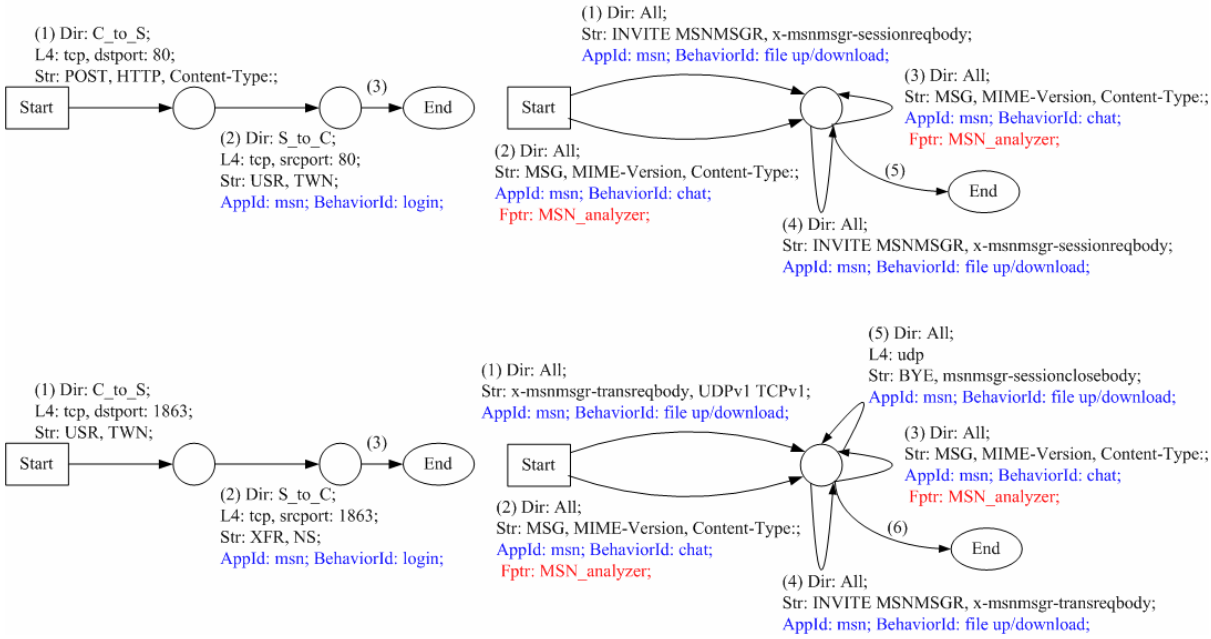

Fig. 6. Two sets of MSN FAs for the official communication and for HTTP tunneling

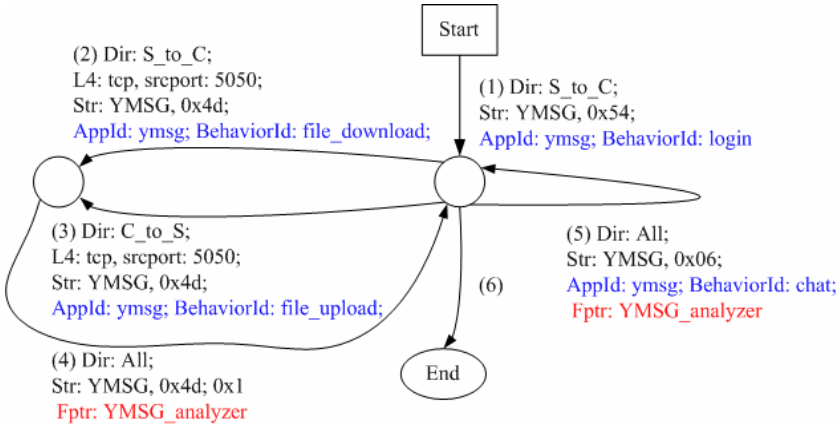

Fig. 7. The YMSG FA for login, chat and file up/download behavior classes

In. Fig. 6, there are 4 transitions to identify every individual activity of delivering chat message. Then, if the keywords are set up, the MSN analyzer receives the full chat contents and searches the strings by regular expression. As described before, the chat messages and files share the same connection between the MSN client and server, so that any behavior blocking will lead to the connection reset. After the first time of file delivering, the client saves the buddy information and then sends the next file in the peer-to-peer mode by UDP directly if possible. For MSN, we use the *transition 5* of bottom right automaton in Fig. 6 to classify these flows. The YSMG system also has a similar peer-to-peer mechanism.

The Yahoo Messenger has a formal 20-byte protocol header and commands in communication can be recognized easily. Unlike the MSN protocol to separate the authentication and instant messaging into two servers, an YMSG client has a permanent messenger server to connect, so there is only one FA for YMSG. The message can be delivered by the messenger server or to the participant directly (if the peer information is already kept in the client and it is also reachable). In the action of file transfer, the YMSG server is not responsible for relaying the data chunks, and only transfers the control message. An active sender gets the IP and port pair of the receiver from an YMSG server, and a passive sender sends its IP and listening port to the receiver via a server. We use the *transitions 4* for invocating the YMSG analyzer to extract IP address and dynamic port to register the oncoming data flow. Like the check in MSN, we are also capable of searching a set of keywords in YMSG messages by the analyzer triggered by *transition 5*.

5.3 Anomaly Detection

The above examples show that the port ranges and strings can model the application and behavior classes. Moreover, our approach can be also applied to the detection of malicious traffic. By using the statistical parameter of repetition frequency, our experiment shows a capability of detecting DDoS attacks which are hardly identified by string matching method. For example, the rule below describes a transition condition of DNS UDP flooding.

```
Dir:C_to_S; L4: udp, dst_port:53, Rep_freq: 500/sec; Action: block.
```

Table 2. The trace statistics of the data sets

Dataset	Start / End	#flows	#packets	Size (MB)
Dorm1	2003-10-16 21:44 / 22:47	1669862	26752972	13933.77
Dorm2	2005-01-07 10:11 / 11:37	213619	10907492	7406.93

Table 3. The detected volume of three methods

Trace	Protocol	port -based (MBs)	siganture (%)	FA (%)	
Dorm 1	FTP	99.454	0.01	active	69.1
				passive	350.8
				total	419.9
	eDonkey	2103.361	89.70	110.9	
	BT	154.681	376.90	376.9	
	MSN	6.469	258.10	258.1	
	YMSG	0.2	103.45	3146.1	
Trace	Protocol	port -based (MBs)	siganture (%)	FA (%)	
Dorm 2	FTP	12.948	0.01	active	29.1
				passive	70.5
				total	99.6
	eDonkey	3.473	91.57	64152.7	
	BT	0.43	1066206.90	1066206.9	
	MSN	67.948	1016.70	1016.7	
	YMSG	0.553	99.70	39130.1	

6 Evaluations

This section reports the experimental results. We evaluate the FA framework with a set of packet traces. First, we compare the detection efficacy of port-based detection, signature matching and protocol analyzer. In this experiment, we focus on FTP, eDonkey and BitTorrent for P2P, and MSN Messenger and YMSG for IM in our analyzed datasets. Second, we report the hottest application activities monitored at a gateway node.

The network traces were captured by the tool tcpdump from multiple links. The traces Dorm1 and Dorm2 were captured from the links at the core of two different universities, and the link speeds are all 100Mbps. The trace Dorm2 was recorded in the situation that the network operator tried to stop the eDonkey and BitTorrent traffic by blocking their well-known ports from the core router. The ServerFarm was collected from a Gigabit Ethernet link between a mail server farm and mail relay servers. This trace is expected to have no IM and P2P traffic and is used to verify the false positives of our methodology. Table 2 details the statistics of packet traces.

We next test the detection performance of three identification methods, including the port-based classification, signature matching and our FA approach (i.e., including the string matching, statistical test and protocol analyzer). The port-based method examines all flows where one of the TCP port numbers is equal to the set {20-21} for FTP, {4661-4665} for eDonkey, {6881-6889} for BitTorrent, 1863 for MSN, and 5050 for YMSG. For signature matching, we followed the same string patterns defined in transition conditions of eDonkey and BitTorrent FAs (but without the protocol analyzers), and for our approach, we use the FAs of FTP, eDonkey, BitTorrent, MSN and YMSG introduced in this paper to perform this evaluation. Table 3 shows the detected volume of port-based method for different protocols, and the ratios of signature matching and FA approach compared to the corresponding port-based detection.

In Table 3, at first it clearly demonstrates a conceivable result that the port-based classification is totally not sufficient to detect the use of dynamic FTP data connections in passive mode. On the other hand, because the signatures we design for FTP aim to identify the FTP control session and its protocol commands, it is reasonable that only an extremely low traffic volume is classified as FTP class only by signature matching both in trace Dorm1 and Dorm2. For P2P, the port-based method completely fails to identify the huge majority of P2P traffic in trace Dorm2 which comes from an environment trying to block P2P traffic by well-known ports. For YMSG identification, we found that the file transfer on non-native ports results in the gap between the port-based/signature method and FA detection. With the great support of YMSG analyzer, the FA method gets a much better detection performance. In summary, in this experiment the port-based identification is reliable enough for the traffic of BT and MSN protocols, but fails completely to recognize the dynamic data connections of FTP and YMSG. This experiment indicates again that the portless and dynamic-data-flow-aware detections for FTP, IM and P2P are imperative and also proves the detection performance of FA approach.

Next, we provide a report from the FA implementation at the gateway node of an organization that has 75 employees and more than 150 endpoint hosts. Table 4 shows the two hottest applications recorded by the system log for 10 days from May 8 – 15, 2006. This statistics give us a preliminary view of what are the most active behaviors in an internal network. We plan to further analyze the behavior distribution over hours of the day, days of the week and month in a large-scale network environment.

Table 4. The statistics of the two hottest applications in an organization

Application	Behavior class	Count
WWW	Http get	10613
	Http post	962
	File download	103
	Cookie	11882
	Java applet/ActiveX	120
Application	Behavior class	Count
MSN	Login	365
	File download	9
	File upload	6
	Chat (client to server)	1682
	Chat (server to client)	1643

7 Conclusion

This paper presents a framework to perform a reliable traffic classification and distinguish activities of specific applications. Our approach uses the flow-based state machine to model a given network application and its behaviors (even with the encryption) and combines the signature matching, protocol analysis and statistical test in order to make use of the strength of three approaches. By keeping the stateful information of Internet traffic, our approach provides a high degree of accuracy on a flow and simplifies the process of feature selection within a packet. We also discuss the system design and the implementation of our framework, including the detection heuristics and system details. The experiment result shows that with the great support of protocol analyzers, the FA method gets a much better detection performance. Finally, by using the statistical parameters, our approach can be also applied to the detection of malicious traffic.

We plan to combine the machine learning technique to our methodology for extracting the signature automatically [13] to improve the manpower-intensive process of FA design. In addition, we are programming the enhancement for analyzers with interfaces for file-related behavior classes to virus detection and file extensions checking. At last, in the near future, we will analyze the application behaviors of a large-scale network environment by an implementation with much better performance.

References

[1] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, “An Analysis of Internet Content Delivery Systems,” *Proceedings of OSDI’02*, pp.315-328, 2002.

[2] S. Sen and J. Wang, "Analyzing Peer-to-Peer Traffic Across Large Networks," *Proceedings of ACM SIGCOMM*, pp.219-232, 2002.

[3] S. Sen, O. Spatscheck, and D. Wang, "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures," *Proceedings of WWW*, pp.512-521, 2004.

[4] T. Karagiannis, A. Broido, N. Brownlee, kc claffy, and M. Faloutsos, *File-sharing in the Internet: A characterization of P2P traffic in the backbone*, Technical report, Available at http://www.cs.ucr.edu/~tkarag/, 2003

[5] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload," *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19)*, pp.314-329, October 2003.

[6] C. Dewes, A. Wichmann and A. Feldmann, "An Analysis of Internet Chat Systems," *Proceedings of the Internet Measurement Conference 2003 (IMC'03*), pp.51-64, 2003.

[7] M. Roesch, "Snort – Lightweight Intrusion Detection for Networks," *Proceedings of LISA 99*, pp.229-238, 1999.

[8] H. Dreger, A. Feldmann, M. Mai, V. Paxon, and R. Sommer, "Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection," *Proceedings of USENIX Security*, pp.257-272, 2006.

[9] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: multilevel traffic classification in the dark," *Proceedings of ACM SIGCOMM*, pp.229-240, 2005.

[10] A. Moore and D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques," *Proceedings of ACM SIGMETRICS*, pp.50-60, 2005.

[11] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Profiling Internet Backbone Traffic: Behavior Models and Applications," *Proceedings of ACM SIGCOMM*, pp.169-180, 2005.

[12] A. Moore and K. Papagiannaki, "Toward the Accurate Identification of Network Applications," *Proceedings of the Passive and Active Measurement Workshop*, pp.41-54, 2005.

[13] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: Automated Construction of Application Signatures," *Proceedings of the 2005 ACM Workshop on Mining Network Data*, pp.197-202, 2005.

[14] S. A. Baset, and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internel Telephony Protocol," *Proceedings of IEEE Infocom*, 2005.

[15] M. Day, J. Rosenberg, *A Model for Presence and Instant Messaging*, RFC 2778, February 2000