P2P Traffic Identification and Control

A THESIS PRESENTED

BY

VIJAYAKUMAR MUTHUVEL MANICKAM

TO

THE GRADUATE SCHOOL

IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

STONY BROOK UNIVERSITY

DECEMBER 2009

Stony Brook University

The Graduate School

Vijayakumar Muthuvel Manickam

We, the thesis committee for the above candidate for the

Master of Science degree,

hereby recommend acceptance of this thesis.

Professor Tzi-cker Chiueh, Thesis Advisor

Computer Science Department

Professor Hussein Badr, Chairman of Thesis Committee,

Computer Science Department

Professor Samir Das, Committee Member,

Computer Science Department

This thesis is accepted by the Graduate School

Lawrence Martin

Dean of the Graduate School

Abstract of the Thesis

P2P Traffic Identification and Control

by

Vijayakumar Muthuvel Manickam

Master of Science

in

Computer Science

Stony Brook University

2009

Since the emergence of P2P networking in the late ‘90s, P2P applications have multiplied, evolved and established themselves as the leading ‘growth app’ of the Internet traffic workload. Studies claim that P2P FileSharing applications are the top bandwidth consuming applications in the Internet overtaking the World Wide Web. Classification of P2P network traffic using only port-based or payload-based analysis have been becoming increasingly difficult with many P2P applications using dynamic port numbers, masquerading techniques and encryption to avoid detection. Therefore techniques which are based on behavioral pattern of P2P traffic are to be identified and adopted.

In this thesis we present the various techniques we devised and implemented for identifying and controlling the bandwidth consumption of P2P download/upload traffic at the Gateway Router entry point of a LAN. Our techniques are a combination of heuristics based on TCP/UDP connection patterns of P2P applications like multiple concurrent TCP/UDP connections, initial packets sizes of P2P connections and payload signature analysis. These techniques have been identified based on the analysis performed on the packet traces of 12 popular Chinese and English P2P applications. We also describe in detail the exact algorithms we formulated combining all these techniques.

A key idea that we identified and implemented is to only control the bandwidth consumed by P2P connections and to not completely block them. This has been seen to be helpful in achieving lower percentage of False Negatives. This is because P2P applications try to workaround the situation if they find their traffic being blocked and adopt new techniques. We used a policy based bandwidth management device called Internet Service Management Device (ISMD) which is typically placed on the LAN side of the Gateway Router such that all traffic between the Router and the LAN hosts pass through it. The techniques identified were implemented as a feature and tested in ISMD leveraging its policy based bandwidth allocation capability. Towards the end we present the evaluation results - False Negatives and False Positives for the 12 P2P applications.

To

My Family

Table of Contents

List of Tables

List of Figures

Acknowledgements

First of all, I sincerely wish to thank Dr. Tzi-cker Chiueh for giving me an opportunity to work on this project and for his guidance and support all through the project. I would also like to thank all the present and past colleagues at Rether Networks Inc. In particular, Dr. Wei Li and Sheng-I Doong, President of Rether Networks, for mentoring me and for their insightful advice. I also want to give my special thanks to my colleague Chia Hao for his help.

Chapter 1

1 Introduction

1.1 P2P Network Traffic

A Peer-to-Peer(P2P) distributed Overlay network is composed of participants that make a portion of their resources like disk storage or network bandwidth directly available to other network participants without the need for central coordination instances like servers or stable hosts. In the case of a P2P filesharing overlay, every peer can simultaneously act as a server and a client, fetching and providing data objects that range from a few megabytes to a gigabyte in size. Also, due to its ease of use and large install base with tremendous amount of shared objects, the peer-to-peer filesharing is one of the most popular applications in the Internet community. Consequently a large portion of the Internet traffic volume is occupied by P2P traffic.

1.2 P2P Protocols

Download of a file using a P2P file sharing application typically involves the following 2 phases:

Signaling phase: a peer searches for the files and determines which peers are eligible to provide the desired file. In many protocols this phase does not involve any direct communication with the peer which will eventually provide the content.

Download phase: The requester contacts one or multiple peers among the eligible ones to directly download the desired file.

Basically P2P protocols differ in the way they have their signaling phase designed which provides a P2P client with the key 'peer' information. First generation peer-to-peer file sharing networks, such as Napster, relied on a central database to co-ordinate look ups on the network. Second generation peer-to-peer networks, such as Gnutella, used flooding to locate files, searching every node on the network. This obviously was not an optimal solution as flooding traffic would congest the network unnecessarily. Third generation peer-to-peer networks use distributed hash tables to look up files in the network. Distributed hash tables store resource locations throughout the network. A major criterion for these protocols is locating the desired nodes quickly.

As an example, in BitTorrent, a client wishing to download a file is supposed to have its corresponding torrent file. Torrent files contain the hash values of a file and the address of a tracker. The client takes the tracker information and hash value of the file from the torrent and establishes a HTTP connection sending the tracker, the file information. The tracker responds with a set of peers which the client tries to contact to start downloading the file. In earlier versions of BitTorrent this initial HTTP message exchanges between the client and the tracker were in plaintext, but the current versions have even these messages in encrypted form including the downloading phase protocol messages. This makes relying only the payload information for P2P traffic classification completely impossible.

The classical traffic classification approach of mapping traffic to applications based on port numbers is now ineffective. This ineffectiveness arises because applications such as network games, multimedia streaming, and Peer-to-Peer file sharing use dynamic ports for communication. Some P2P applications are also masking their identity by using port numbers reserved for other legitimate applications. For example, KaZaA is known to use port 80, which is reserved for Web traffic. An alternative approach is payload-based analysis where packet payloads are searched for characteristic signatures of known applications. Application-layer analysis of packet contents is employed by some commercial bandwidth management tools. This general approach, however, poses several technical challenges. First, these techniques identify only traffic for which signatures are available; maintaining an up-to-date list of signatures is a daunting task. Second, these techniques typically require increased processing and storage capacity. Solutions such as capturing only a few payload bytes are not as effective because many applications intentionally use variable-length padding to obscure application signatures. Finally, these techniques fail to detect encrypted traffic as many P2P applications are now moving towards using encryption.

1.3 Motivation

P2P is often used for illegally sharing copyrighted music, video, games and software. P2P traffic can cause network congestion and performance degradation of traditional client-server applications such as the Web. The legal ramifications of this traffic combined with its aggressive use of network resources has necessitated a strong need for identification of network traffic by application type. This task, referred to as traffic classification, is a pre-requisite to many network management and traffic engineering problems.

1.4 Internet Service Management Device(ISMD)

Internet Service Management Device is a LAN bandwidth management device from Rether Networks Inc., that mainly supports application-level traffic shaping and multi-homing load balancing. Some of its other features are real-time traffic monitoring, historic traffic statistics reporting, worm attack detection/deterrence and firewall filtering.

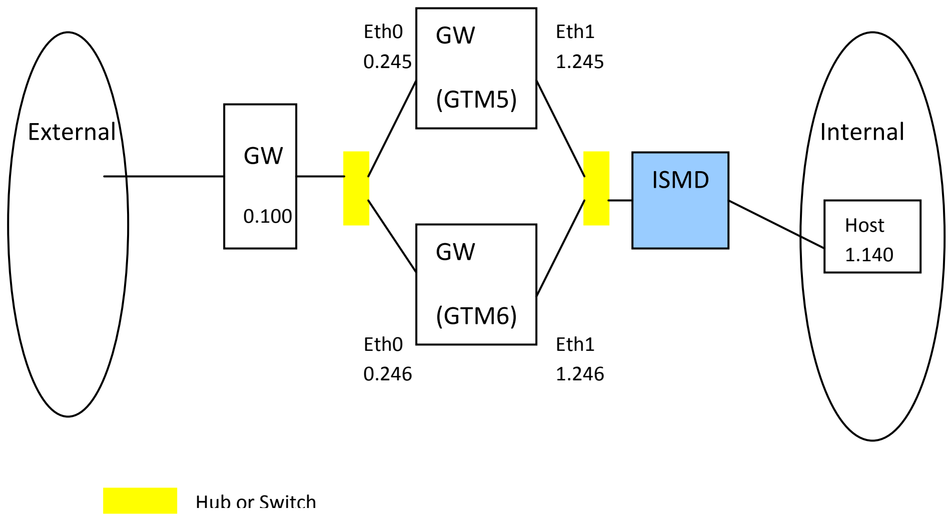

Figure 1 : A Sample Network topology with ISMD

It is designed to work in whichever environment where there is a need to allocate network bandwidth resource based on user-defined policy.

Figure 1 shows a sample network topology to illustrate the position of ISMD in a LAN. ISMD provides a way for the user to set up bandwidth allocation policy in the form of reservations. A reservation can be created by specifying a filter rule and the maximum allowed bandwidth for traffic matching the filter rule. A filter rule can be defined using any of the elements of the four-tuple sourceIP, sourcePort, destinationIP, destinationPort or by selecting an application type. This feature of ISMD's application level traffic shaping has been used for controlling the bandwidth consumed by P2P connections classified based on the implemented heuristics.

Chapter 2

2 Related Work

A great deal of research effort has been devoted to P2P traffic control in the past. Almost all the existing work on this problem targets to detect the P2P traffic at the Internet core (ISPs). In [1] P2P traffic is identified based on the application signatures found in the payload of data packets. Authors showed that typical sets of strings are identifiable in the packet payload generated by some P2P applications. The method can be implemented for online tracking of P2P traffic by examining several packets in each flow. It is reported that the technique works with very high accuracy. However, there are also some drawbacks. The very first challenge is the lack of openly available, up-to-date, standard and complete P2P protocol specifications. Since P2P protocols are continuously developed the traces of today will surely not exist in tomorrow's traffic. Furthermore, an increasing number of P2P protocols rely on encryption, so payload matching cannot be applied in these cases.

A similar payload-based method is presented in [2]. This paper also proposes two heuristics for identification of P2P traffic without payload examination. It is reported that more than 90% of the results provided by the payload method is also identified by the proposed heuristics. The identification of P2P traffic aggregation should be done by heuristics which are based on some common properties of P2P communications instead of examining particular P2P applications.

The method described in [3] also works without payload information. Besides flow identification by ports, it proposes the estimation of unknown traffic by relating it to preceding, known traffic. The authors argue that traffic induces other traffic so there is a possibility to identify unknown traffic which was induced by known traffic. Since this principle cannot guarantee correct identification some additional statistics are also used to increase accuracy in the decision method.

In [4] authors provide a method which is an improvement of the network port-based application detection. Their main idea is to discover the relationships between

flows that belong to a particular P2P application and then use this information to put measured flows into groups. Flow groups together with a set of typical P2P application ports are used to determine whether a group of flows is generated by P2P applications or not. The disadvantage of this method is that it is very difficult to find appropriate typical relationships between flows of a given P2P application. In addition, as presented in the paper, there is still more than 40% of the total traffic
which cannot be identified.

An identification system for pure P2P applications is given in [5]. The method is specialized for the Winny application, which is a most popular P2P application in Japan. It uses the server/client relationships among peers. Some evaluation results of the method are also presented.

Unlike other methods, [6] proposes a multilevel approach of observing and identifying patterns of host behavior at the transport layer. It focuses on associating internet hosts with applications and then classifies their flows. At the *social level* the role of a host is identified in terms of the number of other hosts it communicates with (popularity). It also detects the communities of hosts by identifying and grouping hosts that interact with the same set of hosts. At the *functional level* the role of a host, whether it is a provider or consumer of a service or it participates in collaborative communication (P2P), is determined by analyzing the source ports it uses for communication. At the *application level* the transport layer interaction between hosts along with the knowledge from the previous levels is used to identify the application of origin.

A supervised machine learning approach for P2P traffic identification is presented in [7]. A characteristic library is constructed based on the ratio between the upload and download traffic volume (ud) for the five P2P applications Maze, BitTorrent, PPlive, thunder and eDonkey. The identification mechanism works in two phases. First, the supervised learning phase uses a set of training data to get each P2P application's ud characteristics. Then the traffic identification phase uses these characteristics to determine the application associated with each IP.

In [8] the authors propose a clustering-based framework for classifying network traffic using only unidirectional flow statistics. It uses the following flow features for clustering: total number of packets, mean packet size, mean payload size excluding headers, number of bytes transferred, flow duration, and mean inter-arrival time of packets. The K-Means clustering algorithm is used with Euclidean distance between flow vectors as the measure of similarity.

In contrast to the above methods we present a set of new techniques for identifying P2P traffic at the Gateway entry point of a LAN. We have developed a classifier that uses some of the fundamental characteristics of P2P protocols like the way they create and maintain TCP/UDP connections, some meta information of packets and those based on patterns observed in some of the aggressive P2P applications. The fact that a client has to contact multiple peers with similar packets as soon as it receives the peer information for a file and not all peers are generally up and running is being used. This results in a client trying to establish connections with multiple peers and with only a few successful responses. And the first few initial packets it exchanges with the peers should be similar in size though they may be encrypted.

Chapter 3

3 Packet-Level Characterization of P2P Protocol Implementations

A well known technique, which has been employed in most of the existing commercial products for P2P traffic control, is based on matching the payload signature of P2P applications. The current implementation in ISMD uses a combination of payload signature identification and other heuristics based on the connection patterns observed in networks where the P2P applications are downloading. The techniques which have been identified and used for P2P traffic identification in ISMD are described below.

3.1 Application Payload Signature

P2P applications generally operate based on a proprietary protocol for establishing/maintaining communication between the peers. Often the control messages of these protocols contain a specific signature associated with the protocol, in terms of keywords, commands, options or other easily identifiable bit strings. These bit strings are part of the application layer payload of an IP packet.

P2P application/protocol	String
BitTorrent	0x13"BitTorrent protocol"
	"GET scrape?info_hash="
eDonkey2000	0xe3....0x474601
	0xc5d4....0x405001
Gnutella	"GNUTELLA"
Morpheus	"User-Agent: Morpheus"
Foxy	"User-Agent: Foxy"
Vagaa	"Host: vagaa"

Table 1: P2P protocol payload strings

The presence of such strings in the packet payload can be used reliably to classify those connections as P2P application traffic.This technique has very high accuracy as it can classify P2P traffic deterministically. But at present this technique cannot be depended upon for classifying all of the P2P traffic as many tools have started encrypting their payload messages, in which case signature matching will not work. Table 1 above lists the payload bit string of some of the P2P applications.

3.2 UDP Connection Pattern

Almost all of the P2P applications have been seen to be using UDP connections for some kind of protocol message exchanges, finding the peer neighbors or target file on a peer machine.

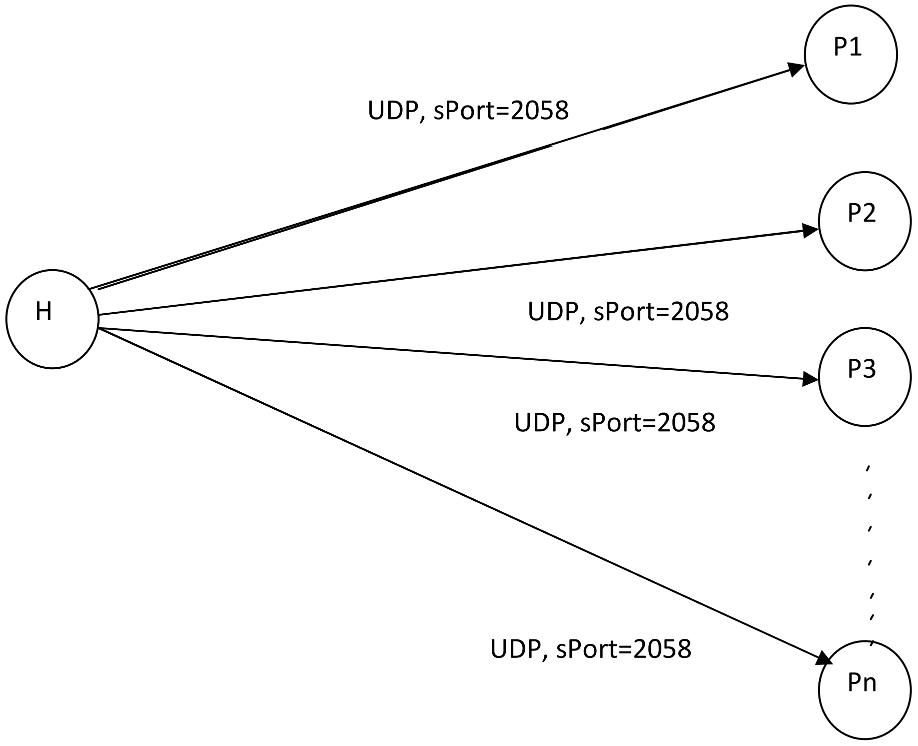

Figure 2: UDP flows using same source port

They keep generating these UDP messages continuously as long as the application is running. A given single instance of a P2P application running in a host uses only a single source port number for all its outgoing connections. This characteristic of multiple UDP connections with same source port from a host is quite different from any other internet application and has been used as an heuristic for identifying the hosts as running P2P applications. This is illustrated in the figure above.

3.3 TCP Connection Pattern

P2P applications first discover the file and file owners by either searching through a centralized server or sending UDP packets to probe the peers. Once they have a set of peers to contact and download from, they send the TCP connection initiation SYN packet to these peers in the form of bursts. Therefore they send out multiple TCP SYN packets to different peers in a short interval of time. And most of the connections use non-standard destination ports as the P2P applications on the peers listen on non-standard ports. This kind of multiple outgoing TCP connections within a short interval of time has been used as an heuristic for identifying TCP connections performing P2P download. Some peers do use standard HTTP ports for bypassing firewalls, but that has been handled with whitelisting as explained later.

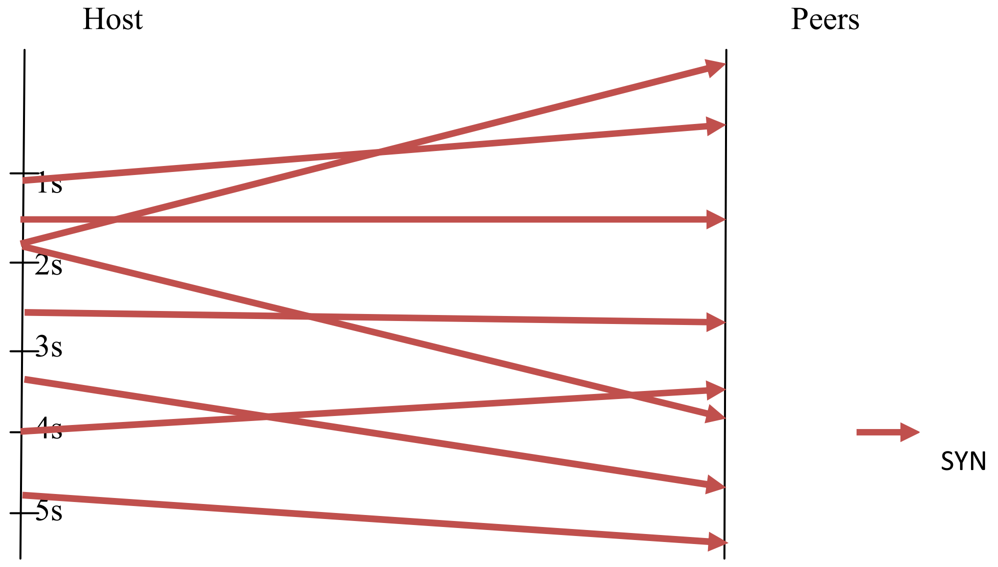

Figure 3: Multiple TCP SYNs sent out in a short interval

3.4 Failed TCP Handshakes

The list of peers that a P2P client discovers for downloading a file only assures that each peer owns a copy of the file but does not guarantee that the peers are currently connected to the internet. Due to this, when a P2P client sends TCP SYN to a set of peers in a short interval, only a few peers respond with a corresponding SYN ACK while other TCP connection initiation handshakes fail. This can be monitored and those successful connections that started along with the failed handshake connections can be identified as P2P traffic.

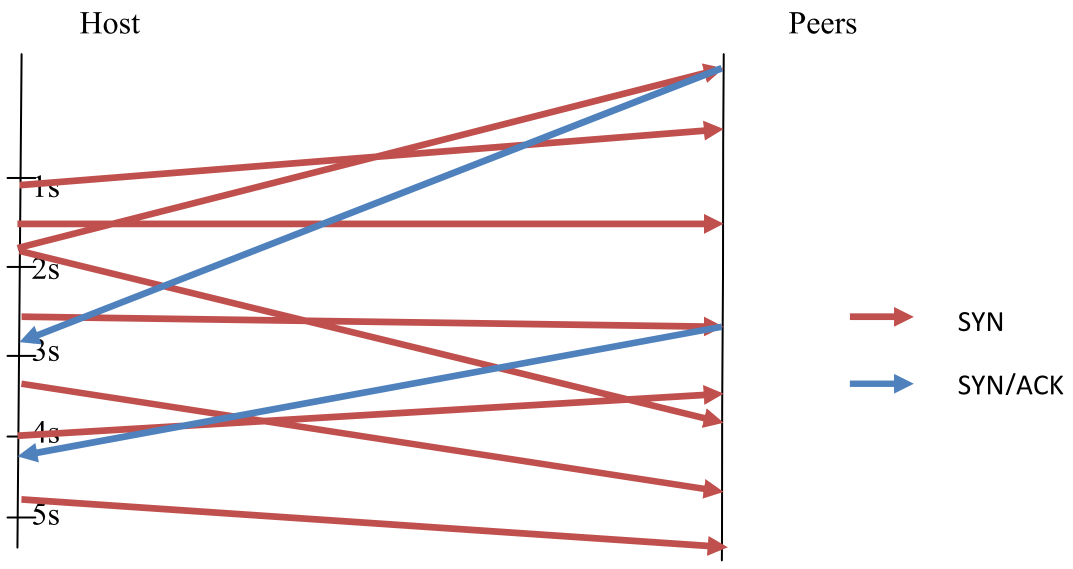

Figure 4: TCP Failed Handshakes

3.5 Abnormal TCP SYNs

It is seen that many of the P2P applications send multiple back-to-back TCP SYN requests to a remote host using the same source and destination ports in a short interval. This is an abnormal behavior characteristic that has been used as an heuristic for identifying such TCP connections that succeed as P2P traffic and the destination hosts as peers.

3.6 TCP/UDP Port Sharing

The single source port number that the P2P applications use for their outgoing UDP connections is also seen to be used by the peers for establishing new incoming TCP connections with the host running P2P. This means that the P2P applications also listen on this port for accepting incoming TCP connections. These TCP connections are seen to be used for downloading and/or uploading. If we keep track of the UDP source ports then all incoming TCP connections with destination port as one of UDP source ports can be marked as P2P connections.

3.7 Initial Packet Sizes of Connections

When a P2P client tries to download a file from multiple peers, it first tries to establish a sTCP connection with the peers. On successful TCP connection creation with some peers it sends certain packets as handshake messages which are specific to the particular P2P protocols used. The packets sent to the peers are all similar since handshake packets remain the same. Therefore the size of these initial packets the P2P client sends to multiple peers remains the same. This has been used as an heuristic to identify such connections which have similar initial packet sizes as P2P traffic.

3.8 HTTP-like Connections

Some P2P applications use the standard HTTP headered (e.g. “GET”) messages for communicating with their peers. Though they use the HTTP headers they do not use the standard HTTP ports 80 or 8080 but some non-standard ports. Assuming that legitimate web-based HTTP traffic should always use a standard port, we can classify these TCP connections as P2P traffic.

Chapter 4

4 P2P Traffic Identification Algorithms:Design and Implementation

Based on the combination of techniques described above, the following design has been adopted and implemented in ISMD. The identification of P2P flows involves two phases.

(1) Identifying P2P suspicious hosts.

(2) Identifying P2P flows in suspicious hosts.

4.1 Identifying P2P Suspicious Hosts

P2P suspicious hosts are identified using the UDP connection pattern technique which is based on the observation of reuse of source port between multiple concurrent UDP connections. The following are the steps involved,

- For each host in the network, ISMD maintains a data structure to keep an entry for the source port number of all the host-initiated out-going UDP connections.
- There is a counter associated with each port entry to keep track of the number of UDP connections that were created and are using that as the source port.
- As new connections are created, if a particular port entry is seen to be used by more than a threshold T1 connections based on the counter value then the host that created the connection is marked as P2P suspicious.
- The counter is also decremented when a connection remains idle for a specified period of time. If the decremented value is less that T1 and if

there are no other ports used by the host whose counter is greater than T1, then the host is cleared from P2P suspicion.

4.2 Identifying P2P Flows in Suspicious Hosts

For identifying P2P flows all the other techniques based on TCP connection patterns, failed TCP handshakes, packet sizes, signatures, HTTP similarity and TCP/UDP port sharing are used. Among these, except for signature based identification which is highly reliable, all the other techniques are applied to only P2P suspicious hosts and also the P2P classification decision in these techniques is made only after a flow is found to be downloading or uploading.

Signature Identification

Signature based method involves classifying flows based on well-known/registered port numbers or matching known bit-strings found in payload of packets. Port based classification is primarily used for whitelisting legitimate applications while payload string matching is used both for whitelisting and identifying P2P flows. The following steps are involved,

- Whenever a new flow is created its initial type is “Unclassified”.
- The new flow is created is monitored for payload string matching for around 10-15 of its initial packets.
- If there is no match based on bit-strings then the destination port of the first packet of the flow is compared with the list of well-known/registered port numbers to classify the flow.
- If there is a match then the flow gets classified as that application type, otherwise it remains as Unclassified.

HTTP-like connections

This is almost part of the signature based identification but it uses some kind of inconsistency detected in the message format of certain flows that use HTTP header bytes in their payload.

- When payload string match is performed on initial packets of flows, some of them match with the HTTP header strings GET, POST.
- If the destination port of the packets carrying these HTTP header strings are found not to match with any of the standard HTTP ports then they are classified as P2P flows.

TCP Connection pattern

- For each new TCP flow initiated by the host a concurrent flow counter is created for recording the number of flows that get initiated with a SYN packet within a short interval T1 around that flow.
- A failed handshake counter is also created which records the count of flows that get initiated within a short interval t1 around that flow but fail to receive a SYN/ACK from the remote host and hence timeout.
- These counters are updated when new flows are created to reflect the count of the number of concurrent and failed flows in that short interval.
- A flow is considered to be downloading/uploading if the sum of its upload and download bytes is greater than 100k.
- If a flow is found to be of Unclassified type and is seen to be downloading/uploading then these counters are checked to decide if it could be a P2P flow.
- If the total number of concurrent flows along the unclassified flow is greater than a threshold T2 and the number of failed flows greater than a threshold T3 then that flow is classified as P2P.

Among the legitimate applications, web browsing HTTP traffic could come closer to having this pattern. Nowadays, many websites are heavy and they initiate more than one simultaneous HTTP connections as soon as their website is requested. But the key thing here is that they would not have any failed connections and so our classifier would not convict them.

<u>Failed TCP Handshakes</u>

- ➢ This is applied for Unclassified download/upload flows.
- ➢ The counter maintained for each flow to record the number of flows with failed handshakes in a short interval around that flow is used.
- ➢ If the number of failed flows around a flow is greater than a threshold T4 (>T3), then that flow is classified as a P2P flow.

<u>Abnormal TCP SYNs</u>

- ➢ When a new TCP flow is initiated by the host with a SYN packet, it is checked to see if there is already an entry for the same flow which is determined by examining the 5-tuple sIP, dIP, sPort, dPort and proto.
- ➢ In case an entry already exists for the flow, then a counter is created for recording the number of duplicate SYN packets associated with that flow.
- ➢ This counter is further updated whenever duplicate SYN packets are seen for that flow.
- ➢ When updating the concurrent and failed handshake counters of a flow, if this kind of flow with duplicate SYNs is encountered then this counter is used for adding more weightage to those counters.

<u>TCP/UDP Port Sharing</u>

- This is applied for Unclassified download/upload flows and the source port counter used for identifying suspicious P2P hosts is used here.
- All UDP flows using a source port whose counter is greater than the threshold T1 are classified as P2P flows.
- If a TCP flow initiated by a remote host is found to be using one of the P2P UDP source ports as its destination port then that flow is classified as a P2P flow.

<u>Initial Packet Sizes of connections</u>

- For all TCP connections the size of first three packets sent by the initiating side are recorded as part of the flow's entry.
- In addition, if the connection is seen to be created along with a set of other connections initiated within a short interval of time, then its initial packet sizes are also stored in a global data structure.
- A counter is associated with each of the packet size entries and is incremented whenever a new connection sends a packet of the same size. This basically records the number of connections that used the same packet size.
- For any Unclassified download/upload flow its initial three packet sizes are checked with the global data structure to see if the counter value for that packet size is greater than a threshold T5. If so the flow is classified as P2P flow.
- For any TCP flow which was whitelisted based on its well-known/registered port number, if all the three of its initial packet sizes are seen to be used by multiple flows based on the counter value for those packet sizes that flow is still classified as P2P flow. This is an heuristic to

identify P2P flows that illegitimately use well-known/registered ports for bypassing P2P detection techniques.

4.3 Whitelisting

The traffic generated by all known legitimate applications is classified based on their well-known/registered port numbers or using well-known protocol signatures. Protocol signature based classification involves payload matching of initial messages with known bit strings like GET/POST for HTTP. The techniques described above for identifying P2P connections are applied only to those connections that remain unclassified after failing whitelisting. But a few techniques with high reliability are also applied to connections that get whitelisted based on only their port numbers. This is because P2P applications intentionally create connections using the well-known port numbers of legitimate applications to bypass the firewalls and P2P detection logic. As port 80/8080 of HTTP is used by many P2P applications for bypassing firewalls a connection is whitelisted as HTTP only if it carries a GET/POST string in its initial messages and uses the HTTP ports.

4.4 Implementation

4.4.1 High-level Data Structures

host_host - This structure maintains the attributes of a host in the LAN for which ISMD is deployed. It stores basic information like IP address of the host and P2P related information like the number of P2P UDP ports (*num_p2p_udp_port)* the host is using, a flag (*is_p2p_host*) to indicate if the host is suspicious of P2P activity at any point in time.

host_hash - This is a hash table of the *host_host* structure which is indexed with the IP address of the host.

lb_flow – This structure is an entry for a TCP or UDP connection passing through ISMD. This stores the basic 5-tuple sIP, sPort, dIP, dPort, proto which uniquely identifies a flow along with statistical information like in-traffic, out-traffic of the flow, start_time etc., It has a pointer(*p2p_info*) to the *stat_p2p* structure and also a field that indicates the application-type (*fl_app*) the flow has been classified to.

flow_hash – This is a hash table of the *lb_flow* structure and it is indexed with the four attributes of a flow sIP, sPort, dIP, dPort.

stat_p2p – This is one of the main structures used for recording the statistical information needed for P2P classification based on TCP connection patterns. It stores information pertaining to a flow and hence is part of the *lb_flow* structure. It has five counters for tracking the number of flows the host initiates in a short time interval around that flow. One counter is used for recording the no. of flows in the same second, two counters are used for recording the no. of flows in the preceding second and two seconds prior to the current flow and similarly two other counters are used for recording the no. of flows that started in the succeeding seconds. It also has a counter for tracking the no. of failed TCP flows around this flow(*n_failed_tcp*), no. of abnormal TCP SYNs(*num_attempts*) for this flow and a pointer to *p2p_pktsz* structure.

p2p_pktsz – This structure also pertains to a flow and it records the size of initial packets of a flow which will be used later for P2P classification criteria if the flow is found to download or upload. This basically contains an array for storing the size of initial packets.

pktsz_hash – This a hash table of initial packet sizes generated by a host and is indexed by packet size and host IP. It has a counter for each entry to track the no. of hits for a packet size. For P2P suspicious hosts, if a flow is seen to be initiated with a few other flows within short interval of time, then for this flow in addition to recording its initial packet sizes in *p2p_pktsz* structure they are also updated in this hash table.

peer_host – This stores an entry for each of the already identified peers and has the peer IP, host IP and a count of no. of times this was identified as a peer on examining the flows for P2P classification.

peer_hash – This is a hash table of *peer_host* structure and is indexed by the peer IP and host IP. Whenever a flow is classified as P2P, the peer IP address from one endpoint of this flow is added in this hash table.

host_port – This structure is used for identifying the P2P suspicious hosts based on reuse of UDP source ports by multiple flows. This stores the port number, the host IP and the count of no. of times a UDP flow has been initiated with this port as source port number.

host_port_hash – This is a hash table of the *host_port* structure and is indexed by the port number and host IP.

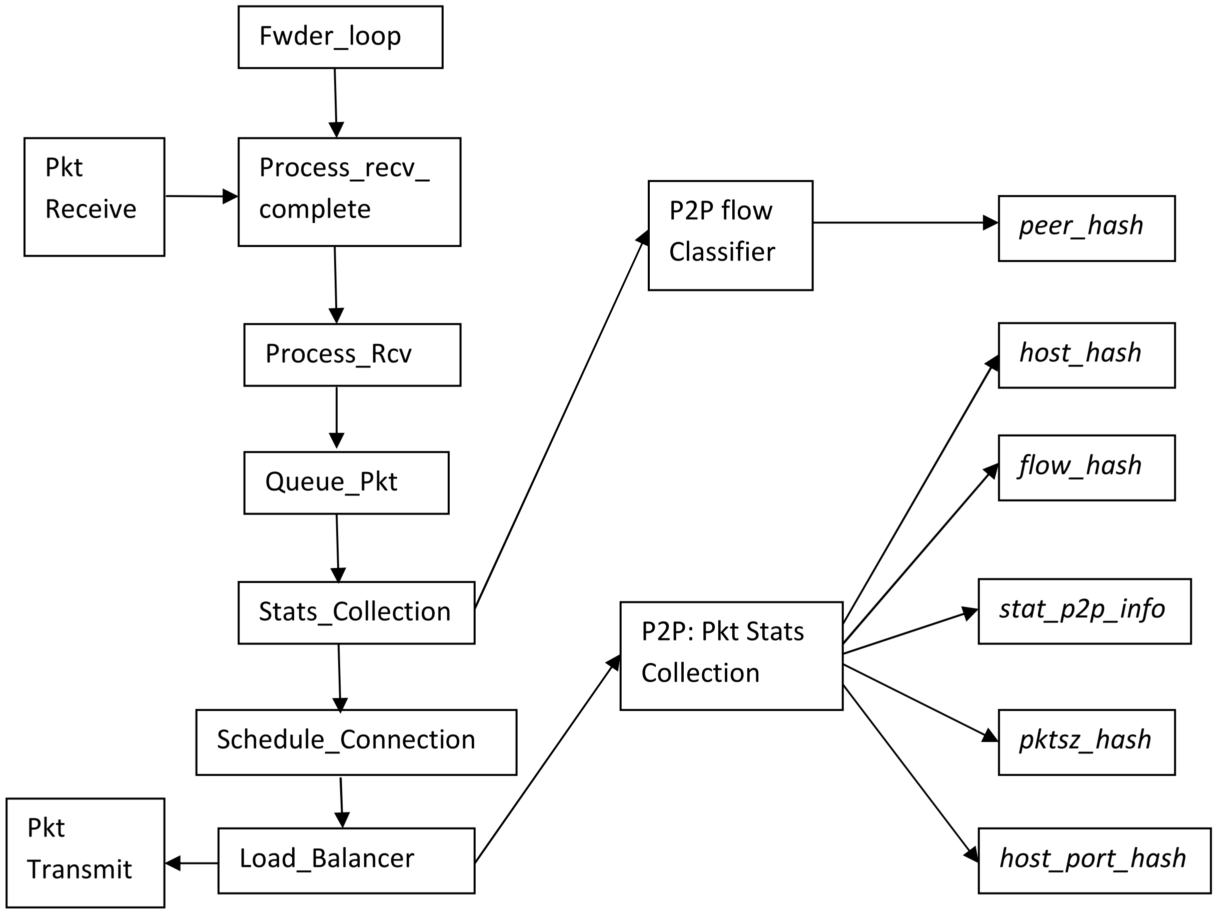

Figure 5 : ISMD Kernel Software Architecture

4.4.2 Functional Implementation of the Techniques

Figure 5 shows the software architecture of ISMD kernel illustrating the addition of P2P traffic identification module and its datastructures. Implementation of each of the techniques inside ISMD kernel are described below.

UDP connection pattern

For any new flow created that passes through ISMD, *host_flow_num_increase()* is called by the load balancing module. This function adds the source port of the flow to the *host_port_hash* if it is not already added else checks if the counter for the port is greater than threshold T1. If so marks that host as P2P suspicious by setting the flag in *host_host* structure. It also increments the *num_p2p_udp_port* counter.

When entry for a flow is removed from *flow_hash, host_flow_num_decrease()* is called by the load balancing module. This function retrieves the port entry for source port of the flow from *host_port_hash* and reduces its counter value. If the counter value reduces below T1, it also decrements the *num_p2p_udp_port* counter. On decrementing, if its value becomes zero then the host is cleared from P2P suspection.

Signature Identification

For every packet passed through ISMD, *stat_comp_app()* is called for updating the statistics information of applications. For any unclassified flow this function calls *stat_match_payload()* for first fiveteen of its initial packet to classify based on rules specified in *sig.bin*. If it is not classified with few of its initial packets then it calls *stat_match_port()* to classify it based on the destination port.

HTTP-like Connections

In *stat_match_payload()* when HTTP header strings GET/POST are matched it checks if the destination port of the flow is any of the standard HTTP ports. If not it classifies that flow as P2P flow.

TCP Connection pattern

For all P2P suspicious hosts when *host_flow_num_increase()* is called for TCP flows, it in turn calls *add_p2p_info()*. This function initializes the counters in *p2p_info* structure of that flow and also updates those counters for all flows that were initiated in a short interval around that flow. It constructs a doubly linked list of *p2p_info* structure of flows belonging to a host in order of their starting time. It traverses this list backwards until it finds flows which were started in time t1 from this flow and initializes the counter that maintains the number of flows created in the same sec(cursec) as this flow and also the counters that maintain those created in the previous secs(precd). While traversing through the list it also increments the counter that maintains the no. of flows created in successive secs(succd) of traversed flows. In this way the cursec, precd and succd counters of all flows are recorded to decide on P2P classification when a flow is seen to be downloading/uploading.

When *host_flow_num_decrease()* is called for flows which have *p2p_info* initialized, it in turn calls *remove_p2p_info()*. This function checks if the flow is removed because of timeout on receiving SYN/ACK. If so it traverses the doubly linked list of *p2p_info* structure in both backward and forward directions and increments their failed_handshake counters to account for this flow's failed handshake.

In *stat_comp_app()* if a flow remains unclassified after applying the signature match and port-mapping it calls *check_app_ppfs()* which checks if a flow is P2P based on the *p2p_info* counters. Using the count of flows initiated in the same sec, previous secs and successive secs it checks if sum of a set of contiguous counters that represent the short time period t1 is greater the threshold T2 and if the failed handshake counter is greater than T3. If so it classifies that flow as P2P flow.

Failed TCP Handshakes

As explained in the TCP connection pattern above the failed handshake counter of flows is updated in *remove_p2p_info()*. In *check_app_ppfs()* if the count of failed handshakes is greater the threshold T4 then the flow is classified as P2P flow.

Abnormal TCP SYNs

Whenever an outgoing packet is sent by a host that passes through ISMD, the function *lb_find_router_of_flow()* of load balancing module is called. This function checks if the flow to which this packet belongs is already added in the *flow_hash* table. If there is an entry for this flow in the *flow_hash* table and if this packet is a TCP SYN, it means that this packet is a duplicate SYN request. The *num_attempts* counter of *p2p_info* is incremented to record this behavior. In *add_p2p_info* and *remove_p2p_info* when traversing the doubly linked list, if such a flow with multiple attempts is encountered then the failed handshake and cursec counter of the flow are incremented *num_attempts* times.

TCP/UDP Port Sharing

As mentioned in UDP connection pattern above, the UDP source ports used by P2P applications are identified by updating the counter in *host_port_hash* table. In *check_app_ppfs()* for incoming TCP flows if the destination port of the flow used by the remote host to connect to the local host is one of the UDP source ports then that flow is classified as P2P flow.

Initial Packet Sizes of connections

The initial packet sizes of outgoing and incoming flows are recorded in the functions *lb_find_router_of_flow()* and *lb_per_session_mac2()* respectively. These functions call *p2p_pktsz_check_add()* which checks if there are atleast T2/2 flows concurrently initiated along with this flow and if so adds this flow's packet sizes to the *pktsz_hash* table. Later whenever a flow is to be examined for P2P classification in *check_app_ppfs(),* the initial packet sizes stored in *p2p_info* of that flow are looked up for in the *pktsz_hash* table. If it is seen that more than T5 flows have used the same packet size then the examined the flow is classified as P2P flow.

4.4.3 Signature and Configuration Parameters Specification

Signature specification includes mapping applications only based on well-known/registered port numbers and a combination of payload bit-strings, port numbers, protocol types like tcp, udp, http etc., Application to well-known/registered port mapping is stored in a file "*app_port_map*" placed in the /root directory. This file is read by the kernel on boot up and it loads them into its data structures.

For bit-string based signature specification the exact string format is specified using regular expressions. The bit-strings, port number and protocol details are specified in human readable format using a kind of meta language and stored in a file "*sig*". There is a user-level program "*gen_sig*" that parses this file, encodes in binary format and stores in a binary file "*sig.bin*". This is again stored in the /root directory and read by the kernel on boot-up to initialize its signature-rule datastructures.

The various threshold values and time intervals mentioned in the udp/tcp and packet size based techniques are specified in a configuration file "*p2p*" which is read by a user-level program "*p2p_conf*" and stored in a binary file "*p2p.bin*". This is also placed in the /root directory and read by the kernel on boot-up into its corresponding datastructures.

Chapter 5

5 Evaluation Results

This section presents the experimental results of the above implementation in ISMD. There are two types of P2P classification inaccuracies, both undesirable:

- Failure to classify the P2P download flows created by P2P applications as P2P, instead classifying them as legitimate application traffic – **False Negatives (FN)**
- The classifier erroneously identifies legitimate application traffic as P2P traffic – **False Positives (FP)**

The accuracy of the classifier was tested using 12 P2P applications which include popular Chinese applications known to be aggressive in creating multiple connections. These applications are based on protocols like BitTorrent, eDonkey2000, Gnutella. Based on the analysis performed on the traffic patterns of these applications a brief description is provided for each of them below.

<u>BitTorrent</u> - As name suggests, it uses the BitTorrent protocol and this was the first client written for this protocol. Its plain text messages can be identified using payload signatures while encrypted flows are identified using concurrent flows and initial packet size similarity.

<u>Vuze</u> - This is one of the latest popular BitTorrent clients and supports protocol obfuscation. It initiates multiple TCP flows in a short time; very few of which succeed, the others failing to receive handshakes.

<u>eMule</u> - This is one of the initial implementations of the eDonkey2000 (ED2K) protocol. It also supports protocol obfuscation and its traffic is identified using the initial packet size similarity and payload signature.

BitComet - This is a BitTorrent client which in addition to protocol obfuscation uses HTTP-like connections to bypass P2P detection techniques. Its traffic is identified using initial packet size similarity and HTTP-like property.

BitSpirit - This is a Chinese BitTorrent client which generates multiple TCP flows aggressively. Its traffic is identified using initial packet size similarity and concurrent TCP flows with failed handshakes.

FlashGet - This is also a Chinese application which uses BitTorrent and other protocols. It mostly connects to port 80 of peers. But its traffic is easily identifiable based on initial packet sizes, concurrent TCP flows and payload signature.

Tuotu - This is a Chinese application which uses both BitTorrent and eDonkey2000 protocols. Its BitTorrent traffic is identifiable using concurrent TCP flows with failed handshakes and ED2K with initial packet size similarity.

Vagaa - This is also a Chinese application using BitTorrent and eDonkey2000 protocols. It initiates a huge number of TCP flows and hence is identifiable using concurrent TCP flows with failed handshakes and also by using initial packet size similarity and payload signature.

Foxy - This is a Chinese application which uses the Gnutella protocol and also generates HTTP-like traffic. Its traffic is identified using payload signature and the inconsistent nature of the HTTP-like connections created by it.

Thunder - This is a Chinese application used mainly for sharing movies, audio, etc. It initiates a huge amount of TCP flows and uses more HTTP connections for download. Its traffic is identified using concurrent TCP flows and initial packet sizes.

DianLei - This is a Chinese application which uses HTTP-like connections for download. Its traffic is therefore identifiable by the inconsistency in HTTP connection.

uTorrent - This is a popular BitTorrent client which initiates multiple TCP flows compared to other English BitTorrent clients. Its traffic are identified using concurrent TCP flows with failed handshakes.

False Negatives

In order to measure the False Negatives, all the P2P clients were installed in a PC connected on the LAN side of ISMD. Each of the P2P clients were run separately for around 4-5 hours and multiple downloads were initiated for each application. All of the traffic that passed through ISMD was that generated by P2P applications. This was ensured by connecting only this PC on the LAN side of ISMD and by running nothing except one P2P application at a time on that PC. A counter was added in the kernel to keep track of the total number of flows created. Another counter was also added to maintain a count of the number of flows classified as P2P by the classification module. The percentage of False Negative is calculated from the difference between these two counters. The table below lists the False Negatives measured for each of the applications.

P2P	Total	No. of flows	False
BitTorrent	2793	2673	4.3
eMule	2509	2407	4.1
Vuze	2992	2815	5.9
BitComet	2809	2604	7.3
uTorrent	2954	2744	7.1
FlashGet	2475	2284	7.7
BitSpirit	2598	2447	5.8
Tuotu	2841	2673	5.9
Foxy	2934	2764	5.8
Vagaa	2791	2565	8.1
Thunder	3219	2920	9.3
DianLei	2839	2689	5.3

Table 2: False Negative percentage for P2P applications

False Positives

In order to measure the False Positives the same setup used for False Negatives was used but running with only legitimate applications. A combination of possible legitimate applications used inside a LAN environment were chosen which would generate traffic somewhat similar to P2P applications. Download Accelerators were run which would create multiple concurrent connections to accelerate the download. Network File Server Traffic were generated which would use both TCP and UDP connections concurrently exihibiting P2P like behavior. Web browsing HTTP traffic were generated automatically using tools like WinHTTrack which would crawl through all urls on a web page and download their contents. Basically this would create multiple flows in short time intervals. The following are the applications that were run:

Download Manager: FDM, Orbit Downloader, Download Accelerator.

Network File Server Traffic: Samba Mount, Windows mapping network drive.

HTTP: WinHTTrack, E-mail sessions

Streaming media : Real Player, Google Videos, Veoh player

Instant messaging: Google Video Chat, Yahoo Video Chat, Skype Video Chat

Macromedia Adobe Flash Player : Yahoo Videos

Miscellaneous : VPN Tunnel traffic, DNS, NetBIOS, WinSCP, FTP

Using the same counters added for measuring False Negatives, the percentage of False Positives was also measured. It was found that out of 7493 flows created by running all these applications 158 flows were identified as P2P by the classification module resulting in a False Positive percentage of 2%. It was seen that these were misclassified as P2P based on the initial packet size heuristic.

Chapter 6

6 Conclusion and Future Extensions

In this thesis we demonstrated the effectiveness of heuristic based techniques when used along with signature based identification for classifying and controlling P2P traffic. These heuristics were based on the behavioral patterns of the traffic generated by P2P applications, such as the way they create and maintain TCP/UDP flows,and the size of intial packets exchanged between the peers. Traffic patterns for a set of 12 popular P2P applications were analyzed to determine these techniques and they were implemented in the Internet Service Management Device. The evaluation results on the 12 P2P applications indicate only a maximum of around 10% of P2P traffic goes undetected with a very low percentage of False Positives.

Some of the limitations of the current implementation and scope for future work include:

- The application of advanced clustering algorithms for identifying the pattern of concurrent flow initiation and matching initial packet sizes for each P2P application could be explored.
- Analyzing the traffic pattern of some more popular P2P applications and tuning the existing implementation to work for them.
- One limitation of the current implementation is that it cannot work accurately if the LAN network of ISMD contains internal LANs with NAT boxes installed. Because in this case ISMD would have visibility of only one 'host' (the NAT box) for the whole internal LAN managed by that NAT box. So even if only one h ost in the internal LAN uses P2P, the traffic from all other hosts in the internal LAN would become suspicious.
- Develop an automated mechanism to come up with a signature for any new P2P application.

Bibliography

[1] S. Sen, O. Spatscheck, D. Wang, "Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures", in Proc. 13th Int. Conf. on World Wide Web, NY, USA, 2004.

[2] T. Karagiannis, A. Broido, M. Faloutsos, K. Claffy, "Transport Layer Identification of P2P Traffic", in Proc. 4th ACM SIGCOMM Conf. on Internet Measurement, Taormina, Sicily, Italy, Oct. 25-27, 2004.

[3] R. Meent, A. Pras, "Assessing Unknown Network Traffic", CTIT Technical Report 04- 11, University of Twente, Netherlands, February 2004.

[4] M. Kim, H. Kang, J. W. Hong, "Towards Peer-to-Peer Traffic Analysis Using Flows", DSOM 2003: 55-67.

[5] S. Ohzahata, Y. Hagiwara, M. Terada, K. Kawashima, "A Traffic Identification Method and Evaluations for a Pure P2P Application", Lecture Notes in Computer Science, p55 Vol. 3431, 2005.

[6] Thomas Karagiannis et al., “BLINC: Multilevel Traffic Classification in the Dark”, ACM SIGCOMM Computer Communication Review, 2005.

[7] Hui Liu et al., “A Peer-To-Peer Traffic Identification Method Using Machine Learning”,Intl. Conference on Networking, Architecture, and Storage, NAS 2007.

[8] Jeffrey Erman et al., “Identifying and Discriminating Between Web and Peer-to-Peer Traffic in the Network Core”, 16th intl. conf. on World Wide Web, 2007.

[9] Marcell Perenyi et al., “Identification and Analysis of Peer-to-Peer Traffic”, JOURNAL OF COMMUNICATIONS, VOL. 1, NO. 7,Nov/Dec 2006.

[10] Michael P. Collins et al., “Finding Peer-To-Peer File-sharing Using Coarse Network Behaviors”, CERT/Network Situational Awareness, CMU.

[11] Chun-Ying Huang et al., “Bounding Peer-to-Peer Upload Traffic in Client Networks”, 37th Annual IEEE/IFIP Intl. conf. on Dependable Systems and Networks DSN'07.

[12] Laurent Bernaille et al., “Traffic Classification On The Fly”, ACM SIGCOMM Computer Communication Review, April 2006.

[13] Jeffrey Erman et al., “Traffic Classification Using Clustering Algorithms”, ACM SIGCOMM 2006.

[14] Fivos Constantinou et al., “Identifying Known and Unknown Peer-to-Peer Traffic”, 5th IEEE Intl. Symposium on Network Computing and Applications, NCA'06.

[15] Azurues Protocol Specification
http://www.azureuswiki.com/index.php/Main_Page

[16] Emule EDKObfuscation
http://mldonkey.sourceforge.net

[17] BitTorrent Protocol Encryption
http://en.wikipedia.org/wiki/BitTorrent_protocol_encryption