Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures

Subhabrata Sen
AT&T Labs-Research
Florham Park, NJ 07932
sen@research.att.com

Oliver Spatscheck
AT&T Labs-Research
Florham Park, NJ 07932
spatsch@research.att.com

Dongmei Wang
AT&T Labs-Research
Florham Park, NJ 07932
mei@research.att.com

ABSTRACT

The ability to accurately identify the network traffic associated with different P2P applications is important to a broad range of network operations including application-specific traffic engineering, capacity planning, provisioning, service differentiation, etc. However, traditional traffic to higher-level application mapping techniques such as default server TCP or UDP network-port based disambiguation is highly inaccurate for some P2P applications.

In this paper, we provide an efficient approach for identifying the P2P application traffic through application level signatures. We first identify the application level signatures by examining some available documentations, and packet-level traces. We then utilize the identified signatures to develop online filters that can efficiently and accurately track the P2P traffic even on high-speed network links.

We examine the performance of our application-level identification approach using five popular P2P protocols. Our measurements show that our technique achieves less than 5% false positive and false negative ratios in most cases. We also show that our approach only requires the examination of the very first few packets (less than 10 packets) to identify a P2P connection, which makes our approach highly scalable. Our technique can significantly improve the P2P traffic volume estimates over what pure network port based approaches provide. For instance, we were able to identify 3 times as much traffic for the popular Kazaa P2P protocol, compared to the traditional port-based approach.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network operations—*Network management, Network monitoring*; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*

General Terms

Measurement, Performance, Design

Keywords

Traffic Analysis, P2P, Application-level Signatures, Online Application Classification

1. INTRODUCTION

Peer-to-peer (P2P) file sharing applications have dramatically grown in popularity over the past few years, and today constitute a significant share of the total traffic in many networks. These applications have proliferated in variety and have become increasingly sophisticated along a number of dimensions including increased scalability, more functionality, better search capabilities and download times, etc. In particular the newer generation P2P applications are incorporating various strategies to avoid detection.

Access networks as well as enterprise networks require the ability to accurately identify the different P2P applications and their associated network traffic, for a range of uses, including network operations and management, application-specific traffic engineering, capacity planning, provisioning, service differentiation and cost reduction. For example, enterprises would like to provide a degraded service (via rate-limiting, service differentiation, blocking) to P2P traffic to ensure good performance for enterprise critical applications, and/or enforce corporate rules guiding running of peer-to-peer. Broadband ISPs would like to limit the P2P traffic to limit the cost they are charged by upstream ISPs. All these require the capability to accurately identify P2P network traffic.

Application identification inside IP networks, in general, can be difficult. In an ideal situation, a network administrator would possess precise information on the applications running inside the network, along with unambiguous mappings between each application and its network traffic (e.g., by port numbers used, IP addresses sourcing and receiving the particular application data, etc.). However, in general, such information is rarely available, up-to-date or complete, and identifying either the applications or their associated traffic is a challenging proposition. In addition, traditional techniques like network port-based classification of applications have now become problematic. Although the earlier P2P systems mostly used default network ports for communication, we have found that substantial P2P traffic nowadays is transmitted over a large number of non-standard ports, making default port-based classification less accurate.

In this paper, we report on our exploration of online, in-network P2P application detection based on application signatures. The following are some key requirements for such an application-level filter. It must be accurate, have low overheads, and must be robust to effects like packet losses, asymmetric routing, etc. (details in Sections 2 and 3) that make it difficult/impossible for a monitoring point to observe all the application-level data in a connection flowing by.

We designed a real-time classification system which operates on individual packets in the middle of the network, and developed application-level signatures for a number of popular P2P applications. Our signatures can be used directly to monitor and filter P2P traffic.

Evaluations using large packet traces at different Internet loca-

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

tions show that the individual signature-based classification (i) has good accuracy properties (low false positives and negatives), even in situations where not all packets in a connection are observed by the monitoring point, (ii) can scale to handle large traffic volumes in the order of several Gbps (GigaBits per second), and (iii) can significantly improve the P2P traffic volume estimates over what pure network port based approaches provide. Our filter has been successfully deployed and is currently running at multiple network monitoring locations.

A lot of existing research on P2P traffic characterization has only considered traffic on default network ports (e.g., [11, 18, 17]). A recent work [12] uses application signatures to characterize the workload of Kazaa downloads. But they do not provide any evaluation of accuracy, scalability or robustness features of their signature. Signature based traffic classification has been mainly performed in the context of network security such as intrusion and anomaly detection (e.g. [5, 4, 19, 14]) where one typically seeks to find a signature for an attack. In contrast our approach identifies P2P traffic for network planning and research purposes. This work, is therefore, more closely related to [8] which provides a set of heuristics and signatures to identify Internet chat traffic. There is also a large body of literature on extracting information from packet traces (e.g., [9]); however, none of these works provides and evaluates application layer P2P signatures.

The remainder of this paper is organized as follows. Section 2 highlights the issues involved in identifying P2P traffic in real time inside the network. Section 3 discusses some of the design choices we made in our approach. Section 4 derives the actual signatures used for P2P detection, and Section 5 describes our implementation of an online P2P application classifier using these signatures. Section 6 presents the evaluation setting, and Section 7 describes the evaluation results. Finally, Section 8 concludes the paper.

2. PROBLEM STATEMENT

We first outline some key requirements of any mapping technique for identifying traffic on high speed links inside the network.

Accuracy: The technique should have low false positives (identifying other traffic as peer-to-peer) and low false negatives (missing peer-to-peer traffic).

Scalability: The technique must be able to process large traffic volumes in the order of several hundred thousand to several million connections at a time, with good accuracy, and yet not be computationally expensive.

Robustness: Traffic measurement in the middle of the network has to deal with the effects of asymmetric routing (2 directions of a connection follow different paths), packet losses and reordering.

The above requirements indicate there are tradeoffs in terms of the level of accuracy, scalability and robustness that can be achieved.

On one end of this spectrum is the current practice of TCP/UDP port number based application identification. Port number based application identification uses known TCP/UDP port numbers to identify traffic flows in the network. It is highly scalable since only the UDP/TCP port numbers have to be recorded to identify an application. It is also highly robust since a single packet is sufficient to make an application identification.

Unfortunately port number based application identification is becoming increasingly inaccurate in identifying P2P traffic. For example, we observed in our traffic traces that a large amount of Kazaa traffic is not using the default Kazaa port numbers most likely — we speculate — to avoid detection.

To address this problem we developed and evaluated a set of application layer signatures to improve the accuracy of P2P traffic detection. In particular this approach tries to determine common signatures in the TCP/UDP payload of P2P applications.

A key challenge in realizing such signatures is the lack of openly available reliable, complete, uptodate and standard protocol specifications. This is partly due to developmental history and partly a result of whether the protocols are open or proprietary. First, the protocols are mostly not standardized and they are evolving. For some protocols (e.g., Gnutella), there exists some documentation, but it is not complete, or uptodate. In addition, there are various different implementations of Gnutella clients which do not comply with the specifications in the available documentation, raising potential inter-operability issues. For a user, this will manifest itself in the form of sometimes poor search performance. For an application classifier to be accurate, it is important to identify signatures that span all the variants or at least the dominantly used ones. At the other end of the spectrum is a protocol like Kazaa, which is developed by a single organization and therefore exhibits a more homogeneous protocol deployment, but is a proprietary protocol with no authoritative protocol description openly available. Finally, just access to the protocol specification is not sufficient - we need signatures that conform to the design decisions outlined above.

Our approach to signature identification has involved combining information available documentation, with information gleaned from analysis of packet-level traces to develop potential signatures. Multiple iterations were used to evaluate the signatures against network traffic data to improve the accuracy and computation overheads.

3. DESIGN CHOICES

Our main goal is to derive application layer signatures for P2P protocols which achieve high accuracy and robustness while being able to apply them at least at Gigabit Ethernet speeds in real time. As we will discuss in Section 7 we achieved these goals by making the following high level design choices.

UDP versus TCP: P2P traffic in principle can flow over UDP and TCP. Since currently most P2P protocols transmitted their data via TCP we focus on signatures found within TCP based P2P traffic. Obviously our signatures could be extended to UDP if so desired.

Packets versus Streams: The P2P application layer signatures can be applied to individual TCP segments or to fully reassembled TCP connection data streams. The advantage of applying them to TCP data streams is that duplicate data has been removed and that signatures can match data which is transmitted in multiple TCP segments. However, the drawback of applying the signatures to TCP data streams is that the TCP segments have to be reassembled in real time on the monitoring device. In our current design we chose to apply the signatures to individual TCP segments which allows us to achieve higher speeds. We therefore focus on developing signatures that do not span multiple TCP packet boundaries. As we will demonstrate we still achieve high accuracy for the 5 applications with the signatures that we develop.

Location of Signature: Again to improve performance we focus on finding signatures which appear in the beginning of the file downloads. Using this approach allows us to focus our signature evaluation on the first few packets of a TCP connection. We will study how many packets our signatures required in Section 7.

Robustness to network effects: We also aim to develop signatures that can independently identify each direction of an application-level communication. This is to enhance the potential of identifying connections for which the filter does not observe one direction of the traffic (due to asymmetric network routing), or misses some signature-carrying packets in one or both directions (caused by either router-based load splitting [16] or other routing instabilities). Independent identification of each direction also serves to decrease the potential of misclassification, by either reinforcing the marking (if both directions identify the same application) or flagging a potential discord (if the 2 directions are identified with different applications). Note that for some usages, such as accounting for total P2P traffic or identifying if some P2P communication is being used, where it is more important to identify that some P2P communications is being used, the last potential (of multiple classifications of the directions) is not an issue.

Early Discard: For efficiency reasons, we shall consider both signatures that identify an application as well as those that indicate that a connection does not belong to an application. The latter category of signatures allows us to quickly identify packets that are not likely application packets, and thereby frees up resources for examining more promising candidates.

Signaling versus Transport: Since the bulk of P2P traffic is related to file downloads and not due to file searches (signaling) we chose to concentrate our efforts on identifying signatures for file downloads rather than the signaling part of P2P protocols.

4. P2P PROTOCOLS AND SIGNATURES

Historically in the client/server model content is stored on the server and all clients download content from the server. One drawback of this model is that if the server is overloaded, the server becomes the bottleneck. The P2P file sharing model addresses this problem by allowing peers to exchange content directly. To perform these file sharing tasks, all popular P2P protocols allow a random host to act as both a client and a server to its peers, even though some P2P protocols do not treat all hosts equally.

Typically the following two phases are involved if a requester desires to download content:

Signaling: During the signaling phase a client searches for the content and determines which peers are able and willing to provide the desired content. In many protocols this does not involve any direct communication with the peer which will eventually provide the content.

Download: In this phase the requester contacts one or multiple peers directly to download the desired content.

In addition to the two phases described above many P2P protocols also exchange keep-alive messages or synchronize the server lists between servers.

In the remainder of the paper we focus on the download phase of the five most popular P2P protocols (*Kazaa, Gnutella, eDonkey, DirectConnect*, and *BitTorrent*). We decided to only track the download phase since it allows us to capture the majority of P2P traffic. We will also only classify the first download in a TCP connection. This simplification is reasonable since it is highly unlikely that two different applications will share a single TCP connection. In the remainder of this Section we will discuss the signatures we discovered for these five protocols. Unless otherwise specified, all the identified signatures are case insensitive.

4.1 Gnutella protocol

Gnutella is a completely distributed protocol. In a Gnutella network, every client is a server and vice versa. Therefore the client and server are implemented in a single system, called *servent*. A servent connects to the Gnutella network through establishing a TCP connection to another servent on the network. Once a servent has connected successfully to the network, it communicates with other servents using Gnutella protocol descriptors for searching the network - this is the signaling phase of the protocol. The actual file download is achieved using a HTTP-like protocol between the requesting servent and a servent possessing the requested file.

To develop the Gnutella signature we inspected multiple Gnutella connections and observed that the request message for Gnutella TCP connection creation assumes following format:

```
GNUTELLA CONNECT/<protocol version string>\n\n
```

And the response message for Gnutella TCP connection creation assumes:

```
GNUTELLA OK\n\n
```

We also observed that there is an initial request-response handshake within each content download. In the download request the servent uses the following HTTP request headers:

```
GET /get/<File Index>/<File Name>
/HTTP/1.0 \r \n
Connection: Keep-Alive\r\n
Range: byte=0-\r\n
User-Agent: <Name>\r\n
\r\n
```

The reply message contains the following HTTP response headers:

```
HTTP 200 OK\r\n
Server: <Name>\r\n
Content-type: \r\n
Content-length: \r\n
\r\n
```

Based on these observations and performance consideration, we recommend the following signatures for identifying *Gnutella* data downloads:

- The first string following the TCP/IP header is ‘GNUTELLA’, ‘GET’, or ‘HTTP’.
- If the first string is ‘GET’ or ‘HTTP’, there must be a field with one of following strings:

```
User-Agent: <Name>
UserAgent: <Name>
Server: <Name>
```

where $< Name >$ is one of the following: LimeWire, BearShare, Gnucleus, MorpheusOS, XoloX, MorpheusPE, gtk-gnutella, Acquisition, Mutella-0.4.1, MyNapster, Mutella-0.4.1, MyNapster, Mutella-0.4, Qtella, AquaLime, NapShare, Comeback, Go, PHEX, SwapNut, Mutella-0.4.0, Shareaza, Mutella-0.3.9b, Morpheus, FreeWire, Openext, Mutella-0.3.3, Phex.

Generally it is much cheaper to match a string with a fixed offset than a string with varying locations. Hence we include 'GET' and 'HTTP' here to help early discard the packets, which do not start with 'GNUTELLA', and also are non-HTTP packets. For robustness, we included the signatures for the request and response header. This way, we can identify Gnutella traffic even if we only see one direction of the traffic.

4.2 eDonkey protocol

An eDonkey network consists of clients and servers. Each client is connected to one main server via TCP. During the signaling phase, it first sends the search request to its main server. (Optionally, the client can send the search request directly to other servers via UDP - this is referred to as extended search in eDonkey.) To download a file subsequently from other clients, the client establishes connections to the other clients directly via TCP, then asks each client for different pieces of the file.

After examining eDonkey packets, we discovered that both signaling and downloading TCP packets have the following common eDonkey header directly following the TCP header:

```
 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8
+-+-+-+-+-+-+-+-+
|     Marker    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               packet Length (4 Bytes)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Message type  |
+-+-+-+-+-+-+-+-+
```

where the marker value is always 0xe3 in hex, the packet length is specified in network byte order and the value is the byte length of the content of the eDonkey message excluding the marker 1 byte and the length field 4 bytes.

Utilizing these discoveries, we recommend the following signatures for identifying eDonkey packets:

For TCP signaling or handshaking data packets, we use two steps to identify eDonkey packets.

- The first byte after the IP+TCP header is the eDonkey marker.
- The number given by the next 4 bytes is equal to the size of the entire packet after excluding both the IP+TCP header bytes and 5 extra bytes.

Since the accuracy for identifying the P2P connections is proportional to the length of the signatures, we tend to include as many fields as we can so long as they do not increase the computational complexity significantly. Here both marker and length fields have a fixed offset, therefore the computational complexity is the same (O(1)) for matching one of them or both, but the accuracy is improved by 2^{32} times compared with matching the marker field alone.

We have also identified the signatures for UDP handshaking messages. However, since UDP is only used for extended searching, and is rare compared with TCP communications, we do not report it in this study.

4.3 DirectConnect Protocol

The *DirectConnect* network is composed of hubs, clients, and a single superhub with multiple servers. All of them listen on TCP port 411 to connect and exchange commands such as search request. Clients (peers) store files and respond to search requests for those files. The single superhub acts as a name service for all the hubs. All hubs register with the superhub and clients discover hubs by asking the superhub. Each of the clients has a username (a.k.a. nick). Normally the clients listen at port 412 for client connections. If the port 412 is already in use, clients will use ports 413, 414 and so on. *DirectConnect* uses TCP for client to server and client to client communication, while UDP is used for communication between servers. The TCP/UDP data is a series of commands or a public chat message. In this study, we focus on the TCP commands. The TCP commands are identified with following form:

```
$command_type  field1 field2 ...|
```

which starts with character '\$', and ends with character '|'. The list of valid command types for TCP communications are: MyNick, Lock, Key, Direction, GetListLen, ListLen, MaxedOut, Error, Send, Get, FileLength, Canceled, HubName, ValidateNick, ValidateDenide, GetPass, Mypass, BadPass, Version, Hello, Logedin, MyINFO, GetINFO, GetNickList, NickList, OpList, To, ConnectToMe, MultiConnectToMe, RevConnectToMe, Search, MultiSearch, SR, Kick, OpForceMove, ForceMove, Quit.

To improve the evaluation performance we evaluate this signature in the following two steps:

1. The first byte after the IP+TCP header is '\$', and the last byte of the packet is '|'.
2. Following the '\$', the string terminated by a space is one of the valid TCP commands listed above.

Although we are matching a list of strings which can be an expensive operation, we shall only perform the string match on packets which pass the first test.

4.4 BitTorrent Protocol

The *BitTorrent* network consists of clients and a centralized server. Clients connect to each other directly to send and receive portions of a single file. The central server (called a tracker) only coordinates the action of the clients, and manages connections. Unlike the protocols discussed above, the *BitTorrent* server is not responsible for locating the searching files for the clients, instead the *BitTorrent* network client locates a *torrent* file through the Web, and initiates the downloading by clicking on the hyperlink. Hence there is no signaling communication for searching in the *BitTorrent* network. To identify *BitTorrent* traffic, we focus on the downloading data packets between clients only since the communication between the client and server is negligible.

The communication between the clients starts with a handshake followed by a never-ending stream of length-prefixed messages. We discovered that the *BitTorrent* header of the handshake messages assumes following format:

```
<a character(1 byte)><a string(19 byte)>
```

The first byte is a fixed character with value '19', and the string value is 'BitTorrent protocol'. Based on this common header, we use following signatures for identifying *BitTorrent* traffic:

- The first byte in the TCP payload is the character 19 (0x13).
- The next 19 bytes match the string 'BitTorrent protocol'.

The signatures identified here are 20 bytes long with fixed locations, therefore they are very accurate and cost-effective.

4.5 Kazaa protocol

The *Kazaa* network is a distributed self-organized network. In a Kazaa network, clients with powerful connections, and with fast computers are automatically selected as Supernodes. Supernodes are local search hubs. Normal clients connect to their neighboring Supernodes to upload information about files that they share, and to perform searches. In turn Supernodes query each other to fulfill the search.

The request message in a Kazaa download contains the following HTTP request headers:

```
GET /.files HTTP/1.1\r\n
Host: IP address/port\r\n
UserAgent: KazaaClient\r\n
X-Kazaa-Username: \r\n
X-Kazaa-Network: KaZaA\r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
```

The Kazaa response contains the following HTTP response headers:

```
HTTP/1.1 200 OK\r\n
Content-Length: \r\n
Server: KazaaClient\r\n
X-Kazaa-Username: \r\n
X-Kazaa-Network: \r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
Content-Type: \r\n
```

For higher Kazaa version (v1.5 or higher), a peer may send an encrypted short message before it sends back above response. Note that both messages include a field called X-Kazaa-SupernodeIP. This field specifies the IP address of the supernode to which the peer is connected including the TCP/UDP supernode service port. This information could be used to identify signaling using flow records of all communication.

Using the special HTTP headers found in the Kazaa data download we recommend the following two steps to identify Kazaa downloads:

1. The string following the TCP/IP head is one of following: 'GET', and 'HTTP'.
2. There must be a field with string: X-Kazaa.

Similar to our Gnutella signatures we include 'GET' and 'HTTP' to early discard non-HTTP packets, so that we can avoid searching through the whole packet to match 'X-Kazaa' if the packet has a low probability to contain HTTP request or response headers.

5. SIGNATURE IMPLEMENTATION

As stated earlier we concentrate on P2P application detection in TCP traffic. In particular we decomposed our P2P signatures into fixed pattern matches at fixed offsets within a TCP payload and variable pattern matches with variable offset within a TCP payload. The fixed offset operation can be implemented cheaply whereas variable pattern matches are substantially more expensive.

To be able to execute the decomposed signatures on real network traffic we implemented them in the context of the Gigascope [7] high speed traffic monitor. In this section we will first discuss the issues involved in evaluating fixed and variable offset signatures and then discuss how we implement them in the context of Gigascope.

5.1 Fixed Offset Match

Implementing a fixed pattern match at a fixed offset within a TCP payload is rather trivial. The complexity of this operation in the worst case is the size of the pattern matched. Despite this simplicity it is useful to provide multiple library functions which perform this operation using slightly different parameters to allow for the easy implementation of diverse signatures. For example, in the context of P2P signatures the offset could be specified from the beginning or end of the TCP payload and the pattern matches could be a byte, a word in little endian byte order, a word in big endian byte order, or a string. Therefore, we implemented a library which provides the following functions:

byte_match_offset: returns true if a byte matches the byte in the TCP payload on a given offset. If the offset is negative it is calculated from the end of the TCP payload.

word_match_offset: similar to byte match offset, except that a word is compared. This function takes as additional argument a flag indicating the byte order of the data in the TCP payload.

string_match_offset: similar to byte match offset, except that a fixed length sequence of bytes (string) is compared.

5.2 Variable Offset Match

There are multiple ways to implement matches at variable offsets in an input stream that involve variable length strings. As discussed in Section 3 we decided to perform the matches on a per packet basis, trading off higher performance against matching strings which span multiple packets.

Using this approach all variable matches we need to perform can be expressed as a regular expression match over TCP payloads. For example, the Gnutella data download signature can be expressed as:

```
'^(Server:|User-Agent:)[ \t]*(LimeWire|
BearShare|Gnucleus|Morpheus|XoloX|
gtk-gnutella|Mutella|MyNapster|Qtella|
AquaLime|NapShare|Comback|PHEX|SwapNut|
FreeWire|Openext|Toadnode)'
```

Due to the fact that it is expensive to perform full regular expression matches over all TCP payloads we exploit the fact that the required regular expression matches are of a limited variety. In particular all of the signatures we need to evaluate can be expressed as stringset1.*stringset2 where stringset1 and stringset2 contain a list of possible strings. This allows us to use the following algorithms for our signatures:

- Standard regex (SR): This is the regular expression match function found in the standard c library on FreeBSD 4.7.
- AST regex (AR): Part of the AST library [10], this code is based on the Boyer Moore string search algorithm [6] extended to handle alternation of fixed strings. To search for an m character long string in a $n \geq m$ character sequence, the Boyer-Moore algorithm has worst case time complexity $O(m + n)$, but often runs in $O(n/m)$-time on natural-language text for small values of m.
- Karp-Rabin (KR): This is a probabilistic string matching technique [13] that compares the hash value of the pattern against the hash value of the sub text of a given search text. The worst case complexity of Karp-Rabin is $O(mn)$, but for many situations is often $O(m + n)$.

5.3 Gigascope Based Implementation

Gigascope is a high speed traffic monitor which can perform a variety of traffic measurement tasks at speeds up to OC-48 (2x2.4 Gbps). To evaluate our signature based P2P classification we included the libraries described above into the Gigascope framework and wrote a set of Gigascope configuration files based on our P2P signatures. In the Gigascope framework these configuration files are translated into C code which is subsequently compiled. The resulting executable is used to perform the network monitoring in real time. Gigascope automatically breaks complex computation into multiple tasks exploiting multiple processors if available. In addition to the real-time P2P detection task we also used Gigascope to collect large datasets for our accuracy evaluation as discussed in Section 7.

When we configured our Gigascope instance we utilized the fact that fixed offset matches are substantially cheaper to execute than variable offset matches. For example, to identify the DirectConnect protocol we need to perform a regular expression match for:

```
types|MyNick|Lock|Key|Direction|
GetListLen|ListLen|MaxedOut|Error|
Send|Get|FileLength|Canceled|HubName|
ValidateNick|ValidateDenide|GetPass|
MyPass|BadPass|Version|Hello|LogedIn|
MyINFO|GetINFO|GetNickList|NickList|
OpList|To|ConnectToMe|MultiConnectToMe|
RevConnectToMe|Search|MultiSearch|SR|
Kick|OpForceMove|ForceMove|Quit
```

However, we also know that the first byte of the DirectConnect TCP payload needs to be 36 and the last byte 124. We therefore configured the Gigascope to only try the regular expression match for DirectConnect if the fixed offset fields match.

Note that we used a similar approach for Gnutella and Kazaa which both use the HTTP protocol for their data transfer. Our setup only performs the regular expression match if the TCP payload starts with GET or HTTP indicating a HTTP payload.

In addition to finding packets which identify a particular connection as belonging to a particular P2P application the classifier also maintains an accounting state about each TCP connection. This accounting state is collected by tracking the TCP handshakes for each TCP connection and accounting all packets with the same IP address port number pairs towards the same TCP connection. A TCP connection record is emitted if either the connection has been closed for two minutes, no traffic has been seen for 8 minutes (the typical TCP keep alive interval) or the record is older than 30 minutes. In particular each connection state contains:

- Byte count in each direction
- Number of packets with zero payload in each direction
- Number of packets with nonzero payload in each direction
- First signature match in each direction

TCP connection reconstruction in the middle of the network is challenging due to the existence of packet reordering, and asymmetric routing, and losses, and the need to scale to large numbers of connections. For our purpose, the more lightweight approach of TCP connection accounting is sufficient. Currently our implementation also only inspects the first fragment of an IP datagram which was fragmented. We could enhance our implementation to reassemble IP fragments, however, it would only provide a marginal benefit since TCP uses MTU discovery to avoid fragmentation. For example, in our experiments we observed less than 0.1% IP fragmentation in TCP traffic.

6. EXPERIMENTAL SETUP

To demonstrate the feasibility of our goal of fast P2P detection using application layer signatures we evaluate our signatures in the three dimensions introduced in Section 2. We evaluate our signature-based classifier in terms of accuracy, robustness and scalability.

6.1 Data Sets

We analyzed two full packet traces from different network vantage points using the Gigascope.

Internet Access Trace: The first trace was collected on an access network to a major backbone and contains typical Internet traffic. The trace covers a 24 hour period on a Tuesday in November 2003 and a 18 hour period on a Sunday in November 2003. The total traffic volume was 120 GB of compressed data and corresponded to 4.58 million TCP connections.

VPN Trace: The VPN (Virtual Private Network) trace was collected on a T3 (45 Mbps) link connecting a VPN containing 500 employees to the Internet. The router on this link blocks P2P ports and corporate policy prohibits the use of P2P applications within the VPN. Therefore, this link has a low probability of carrying P2P traffic. This trace contains 6 days worth of data or 1.8 Terabytes of data in 2.8 billion packets. The data was collected in November 2003.

6.2 Accuracy Evaluation

There are two types of classification inaccuracies, both undesirable

- The classifier erroneously identifies non-application traffic as application traffic. One metric to measure this error is the False Positive (FP).
- The classifier fails to identify application traffic as such. One measure of this error is the False Negative (FN) metric.

Let m denote the total application traffic (total bytes, connections etc.) identified by the signature, and n, the total actual traffic for that application, and t be the total amount of non-application traffic identified as application-traffic. Then the FP and FN ratios are computed as $FP = \frac{t}{n}$ and $FN = \frac{n-m}{n}$. Low false positives and negatives are important to ensure that conclusions drawn from signature based P2P monitoring are accurate.

The misclassification can be caused by various factors including (i) the proposed application signature being too restrictive or too general (identifying only part of the application traffic and/or misidentifying other traffic as the P2P application), (ii) the individual packet-based marking approach could also cause errors as we do not consider any signatures that might be spanning packet boundaries, and finally, (iii) our reconstruction of bidirectional TCP connections from packets could have inaccuracies, e.g., if the timeout used to determine the end of a connection causes a single application connection to be split into multiple connections. In that case, because the application signatures typically occur at the beginning of a TCP connection, our scheme would be unable to recognize some of the split-created connections.

To demonstrate the robustness to some of the network effects discussed in Sections 2–3, we explore the impact on our classification filter to loss of information in one direction. We report the amount of traffic which can be identified based on only one traffic direction of a TCP connection, as a fraction of that amount which can be identified by capturing either traffic direction of a TCP connection.

6.3 Scalability Evaluation

For the application-signature based classification to be usable, the technique has to be able to scale to high speed network links with large numbers of P2P connections. We evaluate the scalability of our technique in two ways:

- *Number of packets to be examined*: We explore the minimum number of packets that need to be checked before we achieve a signature match, for each P2P connection. It is desirable that most P2P traffic can be identified by considering only a very small number of packets at the beginning of each connection.

- *Micro-benchmarking*: The primary component of our signature evaluation is the string search we have to perform on all TCP payloads. We therefore, evaluate the performance of the string search operation using three different algorithms described in Section 5. This evaluation uses the Internet Access Trace and measures the time it takes to process one hour worth of trace data (4 GBytes) for each of the string based signatures introduced in Section 4. The experiments were performed on a Dell Power Edge 2650 with 4 GBytes of RAM and two 2.8GHz processors running FreeBSD 4.7.

7. EXPERIMENTAL EVALUATION

In this section we report our experimental results.

7.1 Accuracy and Robustness Measurements

7.1.1 False Positives

Because of strict firewall restrictions as well as active monitoring and enforcement of corporate rules on use of P2P applications, the VPN Trace offered us a large data set that was expected to contain little or no P2P traffic. We applied our classifier in real time to the VPN traffic to measure the false positive ratio of our technique. Our approach was to investigate manually any TCP connection which is identified as a P2P connection. If in fact the content of the connection did not belong to a P2P protocol we count the connection as a false positive.

Our signatures initially identified 2610 packets as P2P packets out of 2.8 billion packets. We then examined the 2610 packets to determine their status. We identified manually that 82 of these packets contain the string 'X-Kazaa', which is the signature for Kazaa protocol, and the rest of them (2528) contain the string 'BitTorrent Protocol', which is the signature for BitTorrent protocol. We subsequently informed the operator of the VPN about these policy violations. Our signatures, therefore, resulted in at most 2610 false positives out of 2.8 billion (1.8 TeraBytes) and zero false positives assuming that our manual verification is correct.

A second test involved the Internet Access trace. Recall that our approach involved applying each P2P application signature to individual packets, and in case of a match marking that packet as a candidate for that application. We found that there was not a single instant where the same packet was classified as belonging to more than one of the P2P applications. Also there was no case where different packets in the same direction or in different directions of a TCP connection were marked as different applications. This result is important as it suggests that the signatures themselves and the packet-based marking approach were able to unambiguously distinguish between the five different P2P applications.

7.1.2 False Negatives

We next explore the extent of misclassification where the chosen signature fails to identify the application traffic. The Internet Access Trace is used to measure the false negative ratio of our signatures. For this evaluation we assume that all traffic on well-known default P2P port numbers is P2P traffic and that this traffic contains a representative mix of P2P protocol versions. Therefore, any TCP connection on this port number which carries data should be classified as P2P traffic by our signatures and doing so tests our signatures on a representative mix of protocol versions. If a TCP connection on such a port is not identified as P2P traffic we count the connection as a false negative. Note that this approach does not make any assumptions about the traffic on non-P2P ports – in particular we do not assume that it is not P2P traffic.

Alternative approaches for examining false negatives would include getting the traffic dataset by either running each P2P application in an isolated testbed, or running a P2P client and SuperNode locally that communicate with other nodes in the Internet. Given the multiple variants of each protocol, concerns about the need to get packet traces that are representative, and legal issues in actively joining such a system, we adopted the approach of passive monitoring at representative network locations.

We use the following ports [3, 2, 1] for identifying the P2P traffic for this test: Gnutella $(6346, 6347)$, Kazaa (1214), DirectConnect $(411-419)$, BitTorrent $(6881-6889)$, eDonkey $(4661, 4662)$.

Table 1 presents the FN ratio for the different applications. For each application, column 2 presents the total traffic for all connections that use the default application port(s) at at least one endpoint, and column 3 depicts how much of that traffic the signature-based classification missed. We find the FN ratio is less than 5% for all the protocols except BitTorrent for which it is about 10%.

Some of the missed traffic can be attributed to TCP connections where there were no application-level packets exchanged, e.g., for connections with some or part of the SYN-SYNACK-ACK handshake that occurs at the beginning of a TCP connection. The application signature-based classification would miss such connections. To quantify how this might impact traffic accounting, the second set of numbers in each row in Table 1 depict the total traffic based on port numbers and the corresponding FN ratio, when we consider only connections that have at least one packet with application-level payload (data beyond the TCP/IP header) in at least one direction. For Gnutella, it reduces from 4.97% to 1.03%. For each of the other protocols, the FN ratio remains very similar to the "all connections" case. This might indicate that missing the no-application payload connections does not significantly limit the accuracy of the signatures in accounting for the traffic volume.

Some of the missed traffic may be the P2P signaling communications which we excluded from our signature coverage. The FN ratio for eDonkey is significantly lower than Gnutella, Kazaa and BitTorrent, since we included part of the eDonkey signaling in our coverage. Note for DirectConnect, our signatures covered both data and signaling and the FN ratio is almost zero, indicating a near-perfect match. The FN numbers are also upper bounds on the missing signaling overhead.

Finally note that the FN numbers that we present here are estimates, because it is entirely possible that the default port(s) are being used by applications other than the specific P2P protocol. For instance, while BT uses ports 6881-6889, port 6883 and 6888 respectively are also associated with DeltaSourceDarkStar [3], and MUSE. Similarly, ports $411-419$ (default DirectConnect ports) are associated in the IANA database [2] to other applications. Given the low False Positive characteristics demonstrated by the signatures, the FN ratios we obtain above are likely to be overestimates

Protocol	All Connections		Connections with payload	
	Port-based (MB)	FN(%)	Port-based (MB)	FN(%)
Gnutella	486.39	4.97	467.05	1.03
Kazaa	548.41	4.82	547.71	4.70
DirectConnect	2000.75	0.0003	2000.74	0
BitTorrent	54444.67	9.90	54442.37	9.89
eDonkey	2149.84	1.83	2144.92	1.60

Table 1: False Negative for different P2P protocol signatures.

of the actual False Negative if there was only P2P application traffic running on the default ports. Finally, the FN values also provide upper bound estimates on how much other traffic is delivered on the same default ports.

7.1.3 Robustness

We next explore the robustness of our signatures to loss of information in one direction using the Internet Access trace. Recall that this trace was collected at a gateway between the local network and the Internet. We consider two directions: direction corresponds to data being transmitted from inside to the Internet, and the reverse direction. Using auxiliary interface information, we are able to identify for each TCP connection, which packets are transmitted in each of the 2 directions. There are some connections for which we observe only one direction at this link - the other direction being routed (due to asymmetric routing) over some other link. These latter connections are excluded from the study. For this evaluation, we consider only those connections that route both directions of traffic through our monitoring router. We consider the subset of connections that our classifier was able to identify based on signature matches in either or both directions. For this traffic, for each of the 2 directions identified above, we identify the fraction of traffic that is identifiable based on signature matches with only packets transmitted in that direction. We are effectively trying to evaluate a scenario where the connection level summary statistics may be available for both directions (using, e.g., tools like Cisco Netflow [15] which are deployed at routers throughout many ISPs and are more ubiquitous compared to packet monitors), but the packet classification monitor only has access to one direction of the packet data. Table 2 shows the results. We find that whichever direction we select, across all the applications, one-way signature detection is able to detect the vast majority of the connections. This is good news, because it suggests that the signature based detection is quite robust to asymmetric routing which is quite prevalent in the Internet today. This vindicates our strategy of trying to develop signatures for identifying either direction of a TCP connection.

7.2 Scalability

We next consider the following question: *How many application level packets have to be examined on a per connection basis to identify a P2P connection?* For the *Internet Access* data set, for all the connections identified to belong to one of the five P2P applications by our signature-based classification, we compute the minimum number of packets in either direction that are processed before the first successful match with a signature occurs. Fig. 1 plots, for each P2P application, the cumulative percentage of application traffic that was successfully identified based on examining at most N packets, for a range of values of N. The graphs indicate that to identify most of the traffic, for each connection a handful of packets need to be examined before a match is obtained - a welcome news from the efficiency viewpoint.

To evaluate the cost of variable offset matches we measured the performance impact of variable offset matches for the three protocols which require this type of search. In particular we tested the Kazaa, Gnutella and DirectConnect signatures. The signatures in particular contained:

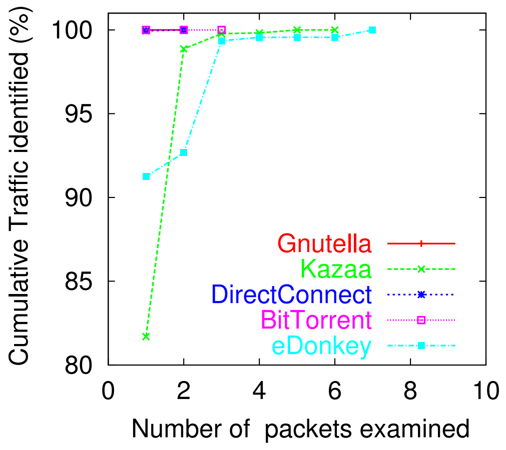

Figure 1: Scalability: The cumulative distribution of total application traffic identified by examining at most N packets, as a function of N.

Kazaa: One 7-character case-insensitive keyword.

Gnutella: Two groups of keywords separated by a random number of whitespaces. The first group contained two keywords combined by a logical OR operation, and the second group contained 17 keywords also combined by logical OR operations. The total size of the regular expression expressing this signature was 178 characters.

DirectConnect: The DirectConnect signature consists of 36 keywords all combined by logical OR operations.

Using the three regular expression algorithms described in Section 5 we measured the bit rate at which we can apply the signatures to all packets using 24 5-minute traces randomly chosen from the Internet Access Trace. The total data in these traces was 4 GB. To contrast the performance of each search algorithm, we first loaded each 5-minute trace once to warm up the file cache, and then measured the time taken to load the trace from the file cache. On our Dell Power Edge with two 2.8 Ghz processors and 4 GB of RAM running FreeBSD 4.7, this resulted in an average data streaming rate of 4.7 Gbps if no search was performed.

Table 3 shows the relative performance of each algorithm compared to the average throughput of just loading the data from a warm file cache. The results show clearly that the Boyer Moor based AST regex search outperforms its competitors by more than one order of magnitude. It also shows that the most complex signature of DirectConnect reduces our system throughput to 8.7%

protocol	Identified Traffic (%)	
	Interface1	Interface 2
Gnutella	99.10	99.99
Kazaa	99.34	99.99
DirectConnect	98.94	95.55
BitTorrent	99.99	99.95
eDonkey	99.69	99.99

Table 2: robustness of signature based identification.

Protocol	libc	AST regex	Karp Rabin
Kazaa	2.39%	77.60%	0.9%
Gnutella	0.27%	58.7%	0.17%
DirectConnect	0.21%	8.7%	0.07%

Table 3: Performance of search algorithms for different P2P protocol signatures relative to the system throughput.

compared to the throughput when no variable offset match needs to be performed. In our system this is still in excess of 400 Mbps. This throughput combined with the cheap filtering provided by the fixed offset matches for DirectConnect is more than sufficient to sustain Gigabit Ethernet links. Both Gnutella (2.8 Gbps) and Kazaa (3.7 Gbps) even perform above the Gigabit Ethernet mark without any fixed offset based filtering. Additional savings can be achieved by only inspecting the first packet as previously shown. This will allow us to perform this type of signature-based traffic identification at very high rates.

7.3 Comparison with port-based identification

As discussed earlier in the paper, the limitations of using port-numbers for application classification was one motivator for this research. Our evaluations above indicate that our identification technique has good accuracy, can scale to large traffic volumes, and, in particular, has very low False Positives. We next use the Internet Access Data set to illustrate quantitatively, how a purely port-based approach would fare against the signature-based identification. Note that the choice of what goes in the list of default ports would impact the above performance. For this experiment, we selected the list of default ports based on information from multiple sources including the IANA database [2], Internet Storm Center mapping [3], and CISCO documentation. The resulting list is identical to the one used for the FN experiments (Section 7.1.2), except that for Gnutella, we add the ports 6348, 6349, 6355, and 5634.

Column 2 in Table 4 presents for each P2P application, the total traffic based on the default ports. Column 3 presents the total P2P traffic identified using application signatures expressed as a percentage of corresponding the value in column 2. The data indicates that for some protocols like Kazaa, Gnutella and DirectConnect, a significant proportion of the traffic is channeled on non-standard ports, and would be missed by the port-based classification. For example, the application signatures identified more than 3 times more traffic than that obtained using the Kazaa port. BitTorrent and eDonkey in contrast seem to be at present mostly using their default ports.

The percentages in Table 4 could also be impacted by the possibility that the numbers in Column 1 also include non-application traffic sharing the default port. If that extra traffic could be identified and removed, the percentages might increase. To remove the effect of such non-application traffic, we next compute the total traffic identified by the application signatures transmitted on non-standard ports, as a fraction of the total application signature-identified traffic. The results are shown in Table 5.

8. CONCLUSION AND FUTURE WORK

In this paper we demonstrated the feasibility, robustness and accuracy of application signature based P2P detection in high speed networks. In particular we described and evaluated the P2P signature of the five most commonly used P2P applications. Our work will directly benefit network operators who have a need to identify P2P traffic today, and researchers who want to accurately study the behavior of P2P networks using data based on accurate application identification in contrast to port number based application classification used in the literature today.

As a more general contribution we evaluated multiple algorithms to perform application layer signature matches, and demonstrated that complex application layer signatures can be evaluated on high speed links.

As we expect that in the future more and more protocols which want to avoid detection will use encryption we believe that application layer signatures will eventually have the same fate as port number based application classification has today. Our future work therefore focuses on exploiting other characteristics of data transfers such as communication patterns, timings and traffic volumes to perform application classification. Additionally we are investigating how to adapt signatures if new protocol versions are introduced.

9. ACKNOWLEDGMENTS

We would like to thank Glenn Fowler for providing us with the AST software library and Phong Vo for valuable discussions on string matching algorithms. We also thank the anonymous reviewers whose suggestions benefited the final version of the paper.

10. REFERENCES

[1] BitTorrent Protocol. http://bitconjurer.org/BitTorrent.

[2] Internet Assigned Numbers Authority (IANA). http://www.iana.org/assignments/port-numbers.

[3] Internet Storm Center. http://isc.sans.org.

[4] P. Barford, J. Kline, D. Plonka, and A. Ron. A Signal Analysis of Network Traffic Anomalies. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2002.

[5] P. Barford and D. Plonka. Characteristics of Network Traffic Flow Anomalies. In *Proceedings of ACM SIGCOMM*

Protocol	All Connections	
	Port-based (MB)	Signature-based (%)
Gnutella	487.12	145
Kazaa	548.41	347.38
DirectConnect	2000.75	163.45
BitTorrent	54444.67	90.97
eDonkey	2149.84	102.37

Table 4: Accuracy of port-based vs. signature based identification.

Protocol	Signature-based	
	Total (MB)	Non-Standard Ports(%)
Gnutella	706.29	34.52
Kazaa	1905.12	72.60
DirectConnect	3270.30	38.82
BitTorrent	49528.90	0.95
eDonkey	2200.92	4.10

Table 5: Fraction of P2P application traffic (as identified by application signature) on non-standard ports.

Internet Measurement Workshop, October 2001.

[6] R. Boyer and J. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[7] C. Cranor, T. Johnson, and O. Spatscheck. Gigascope: a stream database for network applications. In *SIGMOD*, June 2003.

[8] C. Dewes, A. Wichmann, and A. Feldmann. An analysis of internet chat systems. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, October 2003.

[9] A. Feldmann. BLT: Bi-layer tracing of HTTP and TCP/IP. *WWW9 / Computer Networks*, 33(1-6):321–335, 2000.

[10] G. Fowler, D. Korn, and K.-P. Vo. Libraries and file system architecture. In B. Krishnamurthy, editor, *Practical Reusable UNIX Software*, chapter 2. John Wiley, New York, NY, 1995. `http://www.research.att.com/library/books/reuse`.

[11] A. Gerber, J. Houle, H. Nguyen, M. Roughan, and S. Sen. P2P The Gorilla in the Cable. In *National Cable & Telecommunications Association (NCTA) 2003 National Show*, Chicago, IL, June 2003.

[12] 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. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19)*, October 2003.

[13] R. Karp and M. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, March 1987.

[14] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. In *Proc. of the USENIX Security Symposium*, Washington, D.C., August 2001. http://www.cs.ucsd.edu/~savage/papers/UsenixSec01.pdf.

[15] White paper-netflow services and applications. `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.%htm`.

[16] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, October 1997.

[17] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An Analysis of Internet Content Delivery Systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[18] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Marseilles, France, November 2002.

[19] Y. Zhang and V. Paxson. Detecting backdoors. In *Proc. USENIX*, Denver, Colorado, USA, 2000.