

Identifying P2P users using traffic analysis

Yiming Gong 2005-07-21

With the emergence of Napster in the fall of 1999, peer to peer (P2P) applications and their user base have grown rapidly in the Internet community. With the popularity of P2P and the bandwidth it consume, there is a growing need to identify P2P users within the network traffic.

In this paper the author will propose a new method based on traffic behavior that helps identify P2P users, and even helps to distinguish what type of P2P applications are being used.

Current Technology

When it comes to identifying P2P users, currently there are only two choices: port based analysis and protocol analysis. Here is a brief review of both.

Port based analysis

Port based analysis is the most basic and straightforward method to detect P2P users in network traffic. It is based on the simple concept that many P2P applications have default ports on which they function. When these applications are run, they use these ports to communicate with outside. The following is a example list:

```
Limewire 6346/6347 TCP/UDP
Morpheus 6346/6347 TCP/UDP
BearShare default 6346 TCP/UDP
Edonkey 4662/TCP
EMule 4662/TCP 4672/UDP
Bittorrent 6881-6889 TCP/UDP
WinMx 6699/TCP 6257/UDP
```

To perform port based analysis, administrators just need to observe the network traffic and check whether there are connection records using these ports. If a match is found, it may indicate a P2P activity. Port based analysis is almost the only choice for network administrators who don't have special software or hardware (such as an IDS) to monitor traffic.

Port matching is very simple in practice, but its limitations are obvious. Most P2P applications allow users to change the default port numbers by manually selecting whatever port(s) they like. Additionally, many newer P2P applications are more inclined to use random ports, thus making the ports unpredictable. Also there is a trend for P2P applications begin to masquerade their function ports within well-known application ports

such as port 80. All these issues make port based analysis less effective.

Protocol analysis

Despite the poor results found using simple port matching, an administrator has another choice: application layer protocol analysis.

With this approach, an application or piece of equipment monitors traffic passing through the network and inspects the data payload of the packets according to some previously defined P2P application signatures. Many of today's commercial and open source P2P application identification solutions are based on this approach, and include the L7-filter, Cisco's PDML, Juniper's netscreen-IDP, Alteon Application Switches, Microsoft common application signatures, and NetScout. They each do their detection work by doing regular expression matches on the application layer data, in order to determine whether a special P2P application is being used.

Because protocol analysis focuses on the packet payload and raises alerts only on a definite match, any client-side tricks that use non-default or dynamic ports to avoid detection by P2P applications will fail. Using this approach, the result is normally more accurate and believable, but it still has some shortcomings. Here are some points to remember with protocol analysis of P2P networks:

- P2P applications are evolving continuously, and therefore signatures can change. Static signature based matching requires new signatures to be effective when these changes occur.
- With more and more P2P identification and control products on the market, P2P developers tend to tunnel around any controls placed in their way. They could easily achieve this by encrypting the traffic, such as by using SSL, making protocol analysis much more difficult.
- Signature-based identification means that the product should read and process all network traffic, which brings up the issue of how to maintain network stability in a large network. The product may burden network equipment heavily or even cause network failures. If it works inline, what will you do when the product fails?
- Signature-based identification at the application level (L7) is also highly resource-intensive. The higher bandwidth network, the more cost and resources you need to inspect it. Suppose you inspect a 1Gbit or even 10Gbit network link, how much investment must you make to get an appropriate product?

Most importantly, if your organization cannot afford the special appliances or applications that perform protocol analysis, is port matching your only alternative? Fortunately, the answer is no. An approach based on traffic behavior patterns proves to be both functional and cost-effective.

Traffic behavior

Network traffic information can usually be easily retrieved from various network devices without affecting network performance or service availability too much. For small or medium

networks, administrators can rely on their gateway or perimeter equipment logs. For larger networks and ISPs, administrators can enable the Netflow function on their routers or switches to export network traffic records.

Although network traffic information is still coarse in some degree, there is valuable information inside the traffic and useful patterns can be uncovered. Looking at host UDP sessions is one good example of this.

Identifying P2P users

The author of this paper has found that a unique traffic behavior to UDP connection pattern exists with P2P applications. This can be used to process network traffic and find out which hosts are running P2P applications in a decentralized network structure. And all that needed is the network traffic records.

What exactly does it mean to look at a UDP connection pattern, and how can it help us? Before answering these questions, let's review the first popular P2P application, Napster.

Centralized, decentralized and hybrid P2P networks

Napster, written by Shawn Fanning, was first launched in May 1999 and was the first generation of a P2P network. Napster's network structure was centralized, which means it was made up of two elements: central index servers and peers. Central index servers were setup by Napster, which maintained the shared music file information of every online peer. When an active peer wanted to download a music file, it sent an inquiry to Napster's central index server and the latter looked up the request its database and sent back a list of which peers had the desired music files. Then the peer can make direct connection to the peers in the list to get the file.

The network structure of Napster has an Achilles Heel -- it is highly dependent on the static central server. If the central server is down, the network will collapse. This was shown by the actions of the recording industry, which forced the original Napster to be shutdown.

The Napster case illustrates the vulnerability of a centralized network structure and greatly affects the subsequent P2P application. For legal, security, scalability, anonymity and some other reasons, more and more P2P applications nowadays work in a totally or partially decentralized network structure, or are moving in the direction. Major P2P file-sharing networks and protocols, such as Edonkey2k, FastTrack, Gnutella, Gnutella2, Overnet, Kad, all use this concept.

Here the author must make it clear that Bittorrent is not a general purpose P2P network although it is a popular P2P application. It still needs tracker servers; while the network structure of Bittorrent is partially decentralized, the technique discussed in this article can't be used to identify Bittorrent users.

Decentralized means a network structure with no dedicated central index servers. It is a trend for P2P evolution. Today, there are many P2P camps using their own network and

protocol, but normally their network structures are totally or partially decentralized. Some P2P applications such as EMule and Edonkey support fully decentralized protocols such as Kademia, which needs no servers at all. And as a partially decentralized model, hybrid decentralized networks have won broad support from various P2P applications and are thus recognized as the most popular P2P network model.

In a hybrid decentralized network, there are still central servers, but they are no longer dedicated and static. Instead, some peers with more power (CPU, DISK, Bandwidth, and active time) will automatically take over the central indexing server functions, which are called ultrapeers (Supernodes). Every one of them is elected from normal peers and each serves a group of normal peers. They communicate with each other to form the backbone of hybrid decentralized network. New ultrapeers are continuously added when appropriate peers join the network. At the same time, ultrapeers are removed when they leave the network.

In order to join the network, a peer must find a way to connect with one or a few of the live ultrapeers. They get the ultrapeer list by some means such as a bootstrap stored in the program or download from special web site. After connecting to a proper ultrapeer, apart from the normal file transfer work, the P2P application must interact with the P2P network to help them keep connected and live happily in the network, uploading information to the server, checking the status of ultrapeer to which they are connected, getting the most current available ultrapeers, comparing the available ultrapeers situations, actively switching to a better ultrapeer, searching files, probing the status of file suppliers, storing available ultrapeers for future use, and so on. In short, besides the real file transfer traffic itself, peers need to send out many control packets (probe, inform and some other packets) to various different hosts to keep up with the changing network environment in real time. This is the first key element of our traffic behavior identification: *peers need many control purpose packets sent out to interact with the decentralized network during their lifetime.*

UDP connection patterns

Today almost all P2P applications using a decentralized structure have a built-in module to fulfill their interaction work, because there are many control purpose packets needed to be sent out to many destinations. A great deal of the modern P2P networks and protocols select UDP as the carrying protocol.

Why do they select UDP? UDP is simple, effect and low-cost. It does not need to provide guarantee for packet delivery, establish connection, or maintain connection state. All these features make UDP fit for fast delivery of data to many destinations. These are just what P2P applications need. Inspecting different P2P applications carefully, you will find most of the modern decentralized P2P applications adopt a similar network behavior. When they startup, they create one or several UDP sockets to listen, and then communicate with abundant outside addresses during their life by using these UDP ports to assist their interaction in the P2P world. This is the second key element of our traffic behavior identification: *peers keep using one or several UDP ports to make connections to fulfill the control work.*

Now, let's turn to a popular P2P application, Edonkey2000, to see how it can be identified.

Edonkey2000 UDP traffic example

The following is a trace file of Edonkey's outgoing UDP traffic. The output display here is sanitized, so it is only a fraction of the captured traffic. In fact, for this example there were 390 records in just two minutes. For example purposes, the source address is replaced with x and the first column of destination address is replaced with y.

```

11:24:19.650034 IP x.10810 > y.34.233.22.8613: UDP, length: 25
11:24:19.666047 IP x.2587 > y.138.230.251.4246: UDP, length: 6
11:24:19.666091 IP x.10810 > y.127.115.17.4197: UDP, length: 25
11:24:19.681433 IP x.10810 > y.76.27.4.4175: UDP, length: 25
11:24:19.681473 IP x.2587 > y.28.31.240.4865: UDP, length: 6
11:24:19.696907 IP x.2587 > y.162.178.102.4265: UDP, length: 6
.....
11:24:20.946921 IP x.2587 > y.250.47.34.4665: UDP, length: 6
11:24:20.962509 IP x.2587 > y.152.93.254.4665: UDP, length: 6
11:24:20.978275 IP x.2587 > y.28.31.241.5065: UDP, length: 6
11:24:20.993871 IP x.2587 > y.135.32.97.580: UDP, length: 6
11:24:21.009621 IP x.2587 > y.149.102.1.4246: UDP, length: 6
11:24:29.681224 IP x.10810 > y.32.97.189.5312: UDP, length: 4
11:24:29.696903 IP x.10810 > y.10.34.181.7638: UDP, length: 4
11:24:29.716503 IP x.10810 > y.26.234.251.12632: UDP, length: 4
.....
11:26:20.291874 IP x.10810 > y.19.149.0.21438: UDP, length: 19

```

From the output, we can see that all traffic is coming from two source ports, UDP 2587 and UDP 10810 (These ports are randomly selected by Edonkey and the port numbers on different hosts will be different). The destination IP addresses are diverse. In fact, Edonkey uses one port to send out server status requests to the Edonkey servers, and uses another port to make connection, IP query, search, publicize and some other work.

Finding the pattern

A study of some other decentralized P2P applications, such as BearShare, Skpye, Kazaa, EMule, Limewire, Shareaza, Xolox, MLDonkey, Gnucleus, Sancho, and Morpheus leads to a similar result. All these applications have the same connection pattern: they use one or several UDP ports to communicate with many outside hosts during their lifetime. Describing this pattern in the network layer, it can be summarized as:

For a period of time(x), from on single IP, fixed UDP port -> many destination IP(y), fixed or random UDP ports

Experience shows that when x equals five, y equals three, as administrators scanning for a P2P application we will get a satisfying result. Administrators can change x and y values to get more precious or rough result according to their requirement.

In practice, we can export network connection records from corresponding equipment and use a database and shell scripts to process them. For every given minute, if the result shows that any host sends out some number of UDP packets to different hosts from a fixed source port, it is highly probable that the host is a P2P host.

The author of this article setup a test environment on one of China's largest ISP nodes. The network connection records were exported from the router as Netflow data and stored into a MySQL database. With the help of a little script to process all the data, many hosts were identified as P2P peers, and some interesting, locally developed P2P new applications were also discovered.

Dealing with false positives

This sounds like a good method to perform P2P host identification, but what about false positives? Fortunately, this kind of network traffic behavior is seldom seen in other types of usage around the Internet. An exception to this would be if the host is a traditional game server, DNS server or media server. This kind of server will also produce traffic records in which many UDP packets are sent out to many different IP addresses from a single source. But administrators can easily distinguish whether a host is a traditional server because a server normally will not send any kind of traffic on ports other than their functional port, which is not the model used by a P2P host.

The value of this UDP connection pattern is obvious: this approach does not need any kind of application layer information, yet the result is still quite satisfactory. It does not rely on any kind of signatures so newly developed P2P application can still be identified quickly in large networks. Meanwhile, analyzing the network layer information requires almost no extra software or hardware, and dramatically reduces the pressure that might otherwise be put on corresponding equipment.

Disadvantages of this approach

To be sure, this UDP session method also has two disadvantages: it can only be used to identify P2P applications that use a decentralized structure (although most of the modern P2P applications are indeed decentralized). Second, if the P2P application chooses TCP rather than UDP to perform its control function, our identification work will fail.

Identifying P2P applications

Up to this point we have identified P2P users by relying on network connection records. We now go one step further to identify what exactly P2P application a host is running without the help of any high level layer data.

Examining the UDP traffic of different P2P applications more carefully, you will find even more interesting patterns. It has been mentioned that a decentralized network structure needs control purpose packets, and it is not difficult to understand that for a dedicated P2P application, there are many kinds of control packets. Packets of the same control purpose

are very often identical in size. Therefore, the UDP packet can even help us identify exactly which P2P application is running, in the absence of any higher level information.

Most of P2P applications do not have complete documentation on their implementation details and some of them are closed source, so we are still unclear exactly what the makeup is of most applications' UDP packets. Therefore, the author of this article has randomly selected seven decentralized, popular P2P applications and made such observations. The result confirm the hypothesis, that all these applications use some fixed length packets to contact outside.

- **Edonkey2000**

Edonkey2000 uses many 6 byte UDP packets to send out 'server status request'. These kind of packets will mostly be seen when Edonkey launches. Additionally, the packet performing search function is almost always seen, and has a length of 25 bytes.

- **BearShare**

When BearShare launches, it first sends out UDP packets with a length of 28 bytes to many different destinations. Every time BearShare launches a file transfer task, there will be a lot of UDP packets each with a length of 23 bytes, sent out to file suppliers.

- **Limewire**

Limewire uses many 35 byte and 23 byte UDP packets, sent out when Limewire starts. Every time a download task starts, there will be many 23 byte UDP packets communicating with the outside.

- **Skype**

Skype will startup with many 18 byte UDP packets to communicate with the outside.

- **Kazaa** When Kazaa launches, it sends out UDP packet with a length of 12 bytes to many different destinations

- **EMule**

When you start EMule and select a server to get connected, there will be continuously many 6 byte UDP packets sent out to perform 'server status request' and 'get server info'. If you choose to connect to a Kad network in EMule, there will be continuously 27 byte and 35 byte UDP packets appearing in the connection traffic.

- **Shareaza**

During Shareaza's lifetime, you will discover that there are continuously 19 byte UDP packets found in the traffic.

The result of these simple tests is quite interesting. It means that after identifying the peers in the network records, we could use this technology to determine in the future what exactly a peer uses. However, research on the size of different P2P applications' control packets is still in its infant stage and there are many things left to do. For a detailed and accurate result, each application may need special focus and a lot of research work is still needed.

Furthermore, there are other means that can be used and combine with the methods we discussed in this article to better identify P2P users and P2P applications. Some P2P applications will make connections to fixed outside IP addresses to perform such functions as version checks, authentication, downloading bootstrap, or even advertising. For example, Kazaa will connect to ssa.Kazaa.com, desktop.Kazaa.com and some other sites when it operates. Skype will make TCP connection to ui.skype.com whenever it startups.

Also there are other aspects about traffic behavior, such as data transferred. Connection duration may be used in P2P identification but this adds another level of complexity.

Conclusion

As always, there is no one-fit-all solution for the P2P identification work. Although port based analysis and protocol analysis are currently the most important and commonly used technologies, we should not feel content with them. Try a brain head storming, there may be another method cropping up to reinforce the P2P identifies solution.

Acknowledgement

My special thanks to Kelly Martin for his careful review and suggestions!

About the author

[Yiming Gong](#) has worked for China Telecom for more than 5 years as a senior systems administrator, and now he works as a researcher at the Research Department, NSFfocus Information Technology Co.Ltd.

[Privacy Statement](#)

Copyright 2006, SecurityFocus