Measurement and Diagnosis of Address Misconfigured P2P Traffic

Zhichun Li, Anup Goyal, Yan Chen, Aleksandar Kuzmanovic
Northwestern University

ABSTRACT

P2P misconfiguration is a phenomenon in which a large number of peers send P2P file downloading requests to a "random" target on the Internet. This traffic wastes Internet resources and can potentially be exploited for DDoS attacks. On analyzing three Honeynet datasets spanning four years and across five different /8 networks, we found P2P misconfigurations are remarkably prevalent. On an average, they contribute 38.9% of Internet background radiation, increasing traffic by more than 100% every year. This phenomenon includes widely used systems like BitTorrent and eMule as well as systems that are more frequently used in Korea and China.

In this paper, we design the P2P Doctor, a measurement tool, which diagnoses the cause of such events by combining passive monitoring with real-time, active backtracking. Our major findings from the analysis of hundreds of events are as follows: For all the P2P systems, misconfiguration is caused by resource mapping contamination, *i.e.*, the sources returned for a given file ID through P2P indexing are not valid. We find that different P2P systems have different reasons for such contamination. For eMule, we found that the cause is mainly a *network byte ordering problem* in the eMule Source Exchange protocol. The protocol switches the byte order for about 12.7% to 25% of the peers. For BitTorrent misconfigurations, one reason is that the KTorrent implementation of UTorrent peer exchange (PEX) protocol has a byte order problem. The other reason is that anti-P2P companies actively inject bogus peers into the P2P systems using the uTorrent PEX protocol.

1. INTRODUCTION

Peer-to-peer (P2P) traffic has grown to be the dominant source of Internet traffic, surpassing World Wide Web traffic. The most recent Symantec report found that more than 50% of all traffic on the Internet was P2P related [8]. In 2004, an ISP with networks in both the US and Asia reported that P2P protocols accounted for 60% of its traffic in the US and more than 80% of its traffic in Asia [28].

Given the unprecedented amount of P2P traffic flowing through the Internet, misconfigurations, due to bugs or attackers with malicious motives, is a serious problem of both Internet and the P2P systems. It has been reported that attackers are already using P2P bugs for DDoS attacks [7], without compromising a single host.

The first contribution of this paper is to discover the address-misconfigured P2P traffic as one of the major sources of Internet background radiation. Usually it is believed that the so-called "Internet background radiation" [34]—the unexpected traffic sent to all the IP addresses including unused ones—is mainly scanning traffic or DoS backscatter traffic. However, after we analyze the data from three Honeynets/Honeyfarms spanning four years on five different /8 networks, we find that address-misconfigured P2P traffic is one of the major sources (an average of 38.9% in terms of the number of connections) of Internet background radiation. Address-misconfigured P2P traffic is the traffic caused by a large number of peers sending P2P file downloading requests to a target that is not part of the P2P network. In addition, from 2004 to 2007, the address-misconfigured P2P traffic has increased more than 100% each year as shown in Figure 4. Neither academia nor industry, however, are aware of this problem. To the best of our knowledge, we are the first to study the address-misconfigured P2P traffic. We will make such measurement data publicly available for the research community.

We believe that the P2P misconfigured traffic is more prevalent than what we observed for those caused by address misconfiguration. It can also caused by port misconfiguration, firewall or NAT incompatibility. Such problems are rooted from the large number of volunteer-developed P2P software variants, which is often neither well tested nor checked for correctness. Thus they can easily contain subtle bugs. For example, the BitTorrent alone has over 50 different variants [3]. Furthermore, Such traffic can be caused by malicious intent, such from anti-P2P companies. Unlike content poisoning, such unwanted traffic have received little attention.

In this paper, we call for more research in studying such unwanted traffic, and diagnose its root cause. It will benefit both end users and ISPs.

For end users, such unwanted traffic affects their P2P system performance, wastes their machine and power resources, and can involve innocent users into DDoS attacks unconsciously [7]. Furthermore, understanding the behavior patterns of anti-P2P peers will help detect them, and thus evade their tracking to avoid potential warnings and lawsuits. Certainly it is an arm-race and anti-P2P companies can also try to improve their strategies with such knowledge.

Second, from the ISPs' perspective, such misconfigured traffic wastes resources of Internet. As the first step towards studying all P2P unwanted traffic due to software bugs or other misconfigurations, in this paper, we focus on the traffic caused by address misconfiguration. Such traffic is not confined to unused IP address destinations. As a first order estimation, after linearly extrapolating the address misconfigured P2P traffic observed in our datasets and applying it to the whole Internet, we find it consumes modest bandwidth, 7.9Gb/s globally. Moreover, this traffic is mostly intercontinental. For example, The sources of such traffic for two US Honeynets are mostly in Asia and Europe, as discussed in Section 2.3.7. Therefore, given a 10Gb/s intercontinental link (*e.g.*, between LA and Tokyo) costs *$1.4 million* per year to lease [2], ISPs might want to remove such traffic for a more "green" Internet.

The second contribution of this paper is to design schemes for detection and root cause diagnosis of address-misconfigured P2P traffic.

Based on the motivations above, we believe that we need to build tools to facilitate the *detection* and *diagnosis* of such unwanted traffic. Here detection refers to identifying the class of misconfiguration scenarios, *e.g.*, the variant and version of particular P2P software that contain bugs, or events caused by anti-P2P companies. Root-cause diagnosis means to locate the software code which contains the bug or anti-P2P peers which trigger the unwanted traffic.

As the first step, we focus on detecting and diagnosing the address-misconfigured P2P traffic in this paper. Again, we are the first one on these efforts, to the best of our knowledge. We design the following general principles that can be applied to detect other misconfiguration/bugs: (1) testing in a controlled environment and tracking the information flow, and (2) combination of passive monitoring and active backtracking to identify misconfiguration events in the wild. The first component can also be applied to build a third party facility to test and verify P2P software which is completely missing nowadays. It will help improve the design, implementation, and misconfiguration-resistance of P2P software. For example, in our study, we found there is a big variation for address misconfiguration resistance for different BT software. Such testing and verification can also help users to select better P2P software.

Applying the detection principles above, we design and implement *P2PScope*. Our analysis shows that *all* address misconfigurations share the common problem of resource mapping contamination, *i.e.*, the peers returned through indexing for a given file ID are invalid, thus causing data plane traffic radiation. Different systems have different reasons for contamination. We performed root cause analysis for two largest P2P systems [28]: eMule and BitTorrent.

For eMule, One major root cause is a network byte ordering problem in the source exchange protocol. The client attempts to contact an eMule client at an IP address with bytes in reverse order from the intended IP address. We confirmed aMule (an eMule variant) has the byte order bug. We also estimated peers with reversed byte order involved in these events to be between 12.7% and 25% out of total peers in the events.

For BitTorrent, address misconfiguration is mostly disseminated by the Peer Exchange (PEX) protocol implemented by uTorrent-compatible clients. We find multiple reasons behind BitTorrent misconfiguration. Some anti-P2P companies actively inject bogus peers through the uTorrent PEX protocol using a modified Azureus client. Furthermore, KTorrent has a byte ordering problem in the uTorrent PEX protocol implementation, which causes the misconfigurations. We have confirmed the bug in the source and with the developers.

Furthermore, we find some unique characteristics of anti-P2P peers, which can be used to detect anti-P2P peers. We do not observe any false positives when using such detection mechanism comparing with existing anti-P2P blacklists.

This paper is organized as follows. In Section 2 we discuss the prevalence of address misconfiguration in time and space and other characteristics. In Section 3, we present the design of our P2PScope system. The root cause inference system and the analysis of address misconfiguration are presented in Section 4. We discuss related work in Section 5 and conclude in Section 6.

2. DEFINITION AND OBSERVATION WITH PASSIVE MONITORING

2.1 What is Address-Misconfigured P2P Traffic?

P2P address misconfiguration is the phenomenon in which a large number of peers send peer-to-peer downloading requests to a "random" target on the Internet. We call such traffic address-misconfigured P2P traffic. We detected such traffic via a set of Honeynet systems that monitor unused IP space. We expect to observe only background radiation. Normal P2P traffic should not be seen in these Honeynets. Such systems employ low- or medium-interaction Honeypots [30], which provide "fake" responses of varying detail, and thus can elicit additional activity from the sender. A large number of peers believe that the resources they need reside at an unused IP and thus direct a large amount of traffic towards it. This IP, however, is part of a large block of IPs that have been non-functional for years. It is extremely unlikely that an unused IP has been a peer in a P2P system. Moreover, because the IP is unused, it is unlikely that the communication with the IP is an intentional DoS attack.

Usually, a given IP in Honeynet is targeted by address-misconfigured P2P traffic from all over the world and that lasts from a few hours to a few months. We call such an *event* an address misconfiguration event. Such events can be triggered intentionally or unintentionally, as discussed below. In this paper, we use "address misconfiguration" to refer to both the cases unless denoted otherwise.

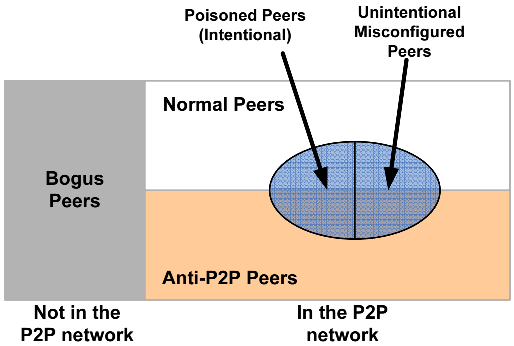

Figure 1: Peer classification

2.2 Definition and Classification of Peers

A P2P network is comprised of different kinds of peers. To ease the discussion, we classify peers based on their characteristics and functionalities. Below, we present the major groups, also shown in Figure 1.

Normal peers: Peers which are in the P2P network. They join the P2P network and conform to the P2P protocol.

Anti-P2P peers: Peers which belong to anti-P2P companies. While such peers are typically blacklisted by a subset of other peers, they are still in the network.

Bogus peers: Peers that are not in the P2P network, such as IP addresses in the Honeynets. Still, such peers are contacted by misconfigured peers.

Misconfigured peers: These peers are a subset of normal peers or anti-P2P peers which are misconfigured and thus try to connect to bogus peers.

Poisoned peers (intentionally misconfigured peers): Misconfigured peers affected by index poisoning [24].

Unintentionally misconfigured peers: Misconfigured peers which are not related to poisoning or any kinds of attacks. They are victims of software bugs or unintentional misconfiguration.

2.3 Passive Measurement Analysis

Surprisingly, we find that P2P address misconfigurations are not confined to a single type of P2P system. We have observed it across numerous different P2P protocols, including the most popular P2P systems: BitTorrent, eMule and Gnutella. Other instances occur in systems that are used only in certain countries (*e.g.*, Korea or China) which differ from those hosting the Honeynets. We also find that the address misconfiguration traffic is increasing over time. Therefore, this problem will be more serious in future. It is also widely spread in the IP space, *e.g.*, across the Honeynet sensors of five different /8 networks.

2.3.1 Data Collections

In our study, we mainly use three datasets collected at different Honeynet/Honeyfarm sensors with different /8 networks and configurations.

The LBL Honeynet dataset The LBL Honeynet sensor is in the Lawrence Berkeley National Lab with five continuous /24 IP blocks in one /16 network with the Honeyd [30] configured. We have data from 2004 through 2007. It helps us understand how the P2P address-misconfigured traffic changes over time. Honeyd is configured such that it will simulate the common protocols on well-known port numbers and act as an echo server for the remaining port numbers. Most P2P protocols are quite symmetric and echo server can simulate the initial handshaking with the P2P peers. In 2008, the traffic collection of the address-misconfigured P2P traffic has been stopped, because DMCA repeatedly complains the Honeynet IPs hosting their movies (actually false positives).

	LBL	NU	GQ
Sensor size	5 /24	10 /24	4 /16
Trace size	901GB	916GB	49GB
Starting time	2004/03/07	2006/09/01	2006/01/03
End time	2008/01/21	2007/12/31	2006/01/28

Table 1: Data Description

The NU Honeynet dataset The NU Honeynet sensor is in Northwestern University and has 10 discontinuous /24 IP blocks within three different /8 IP prefixes. It is also Honeyd configured. It used the same Honeyd configuration as LBL Honeynet until 08/23/2007. After that we developed the P2P-enabled Honeynet for the P2PScope system, which simulates P2P protocols on all port numbers. For the details of the P2P-enabled Honeynet, please refer to Section 3.2.1.

The GQ Honeyfarm dataset We have 26 days of traffic from GQ honeyfarm [15] at International Computer Science Institute, which was originally designed to capture worms and botnets. The GQ honeyfarm has four /16 networks in one /8 network.

2.3.2 Address Misconfiguration Identification

We first separate the scanning traffic which is caused by botnets or worms from the misconfiguration based on whether they have malicious payloads and/or whether they scan a large number of IP addresses (≥ 10) in the Honeynets. We have also checked the traffic remove against P2P protocol signatures, and have not found any matches.

For remaining traffic, we detect P2P connections based on whether their payloads match a P2P protocol signature. For signature matching, we have enhanced the P2P signatures from the L7-filter [1] and integrated it with Bro's dynamic protocol detection framework [17, 29]. We find that no external peers contact more than four IPs in a Honeynet per day. In 97% of cases, peers only contact one Honeynet IP per day. Therefore, we group the P2P connections into address-misconfiguration events based on their target destination. We mainly analyze the events with more than 100 unique sources seen in a six hour interval. For smaller events it is hard to profile their patterns. The event time boundary is identified when the number of unique sources in a time interval is less than one-third of the peak number of unique sources.

One problem with the method is that it only conservatively estimates the lower bound of the number of P2P connections and the number of events seen in the Honeynets. We address this by adding a "Likely" category in Table 2 for events in which we cannot identify which protocol is being used, but its traffic pattern is similar to an address-misconfiguration event. These "likely" events follow the same pattern — a large number of sources contact a few hotspots. We conjecture these are actually address-misconfigurations events, because there are no similar connection patterns known for Internet background radiation. For the events with more than 100 unique sources seen in a six hour interval, the likely cases account for only 15% of the total cases, showing we are able to identify most cases accurately. We have manually checked the payloads of a few "likely" cases, and have found the format of packets are similar to other packets in the same event. We cannot, however, identify the protocol.

2.3.3 *P2P System Diversity*

Table 2 shows the percentage of P2P connections which match P2P signatures and the number of address-misconfiguration events found in LBL and H2 Honeynets. Each row of the table represent a P2P protocol. The "other P2P" category is for connections that matched P2P protocols other than the six protocols listed in Table 2.

We find that address-misconfiguration events mainly come from six different P2P systems. BitTorrent and eMule are the most popular P2P systems, and thus also generate maximum traffic and events. We also find a number of address misconfigurations from Gnutella, Soribada, Xunlei and VAgaa.

BitTorrent is one of the most widely used P2P protocol for distributing large amount of data. More than 50 different BitTorrent clients exist. eMule is the open source alternative to eDonkey and has richer functionality than eDonkey, which is now obsolete. eMule is popular in Europe. Most people use eMule and its variants including aMule, xMule and eMule Mods (modifications of the original eMule). Since they support almost the same protocols and even share some code base, in this paper we refer to all the variants as eMule. Gnutella is another famous decentralized unstructured P2P systems. Soribada is a P2P system which is popular in Korea. Xuelei and VAgaa are two popular P2P systems in China.

These P2P systems include centralized, decentralized DHT-based, and decentralized unstructured P2P systems. This suggests that the prevalence of address misconfiguration is not confined to a single type of P2P system.

	event ($\geq$ 100 uniq srcs)		P2P connections by payloads	
	LBL	NU	LBL	NU
eDonkey/eMule	143	416	5.96%	76.52%
BitTorrent	74	211	75.1%	19.79%
Gnutella	4	3	2.48%	3.65%
Soribada	6	0	15.8%	.0001%
Xunlei	12	0	0.34%	0%
VAgaa	1	1	0.14%	0.013%
Other P2P	0	0	0.17%	0.018%
Likely	73	20	N/A	N/A

Table 2: Address-misconfigured P2P traffic distribution and event distribution (LBL and NU Honeynet datasets).

2.3.4 *Characteristics of Address Misconfiguration Events*

We also find such events are quite heterogenous. The scale of events (in terms of duration and the number of unique sources involved) varies substantially.

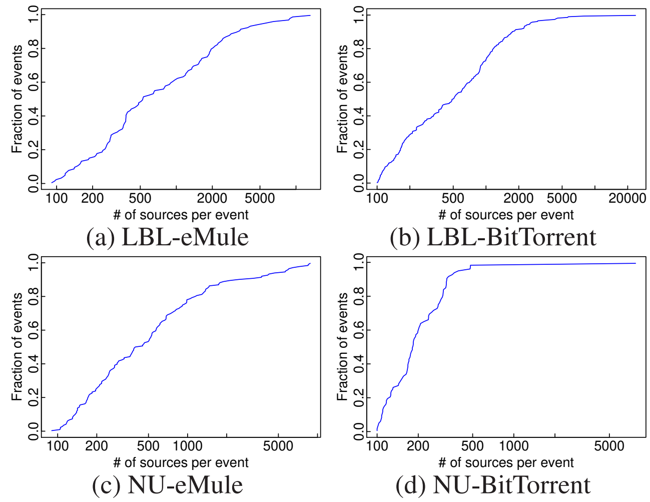

Figure 2: CDFs of events in terms of number of incoming peers.

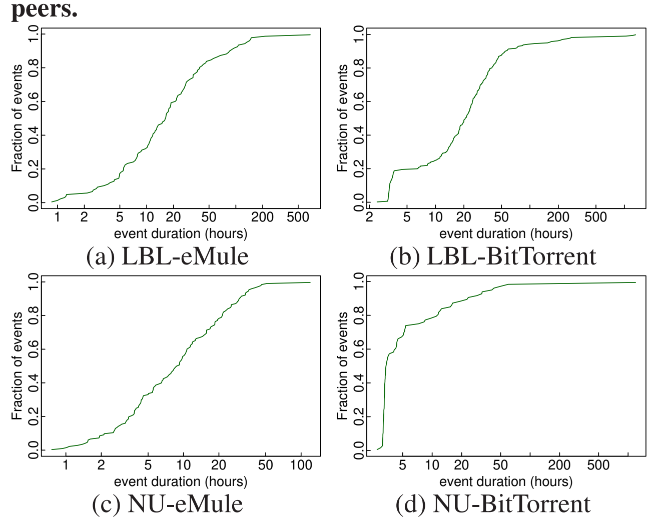

Figure 3: Event duration distribution.

Figure 2 shows the scale of the events in terms of number of incoming misconfigured peers. In all the four cases we find the scale of the events change dramatically. The means of the number of peers in eMule events are 1,404 for LBL and 1,028 for NU. BitTorrent events have smaller scale. The means of the number of peers in BitTorrent events are 894 for LBL and 291 for NU.

Figure 3 shows the ditributions of duration of events. Some events are as short as 1 – 3 hours; some are as long as one month. From Figure 3 (b) and (d), we notice that a group of short events with similar duration. Further study shows those events are anti-P2P peers dominated events. More details regarding anti-P2P related discussion are in Section 4.3.1.

2.3.5 *Prevalence in Time and Space*

As shown in Table 1, we have LBL data spanning four years — 2004 through 2007. We use this data to analyze the temporal trend of address-misconfiguration. As we mentioned, the scale of the events can differ substantially. There-

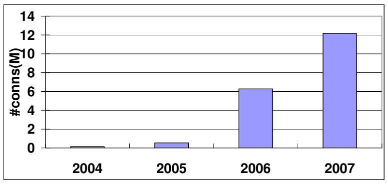

Figure 4: The total number of connections which match the P2P payload signatures from 2004 to 2007 (LBL).

fore, instead of using the number of events we use total number of connections which match the P2P signatures in a year as the metric for studying temporal trend. This metric gives us the lower bound of the number of P2P connections the Honeynet received. Figure 4 shows the increasing trend of address-misconfigured P2P traffic. We also studied the trend on smaller scales (days and months), but due to the large fluctuation of P2P address misconfigured traffic the trend is less obvious. Additionally, we analyzed the annual trend of the total volume of the Internet background radiation in the LBL sensor. We find the yearly total number connections from 2004 through 2007 is: 126M, 28M, 22M and 33M. The large volume of background radiation in 2004 is due to the aftereffects of the Blaster and Welchia worms. For the remaining years, the volumes remain more stable when compared to the growing trend of the misconfiguration traffic.

We also find that address misconfiguration is prevalent across the IPv4 space. As we mentioned earlier, the LBL and NU Honeynets are in four different /8 as well as in /16 networks. In all the four /8 prefixes, we find misconfiguration traffic. We also observe the similar behavior in the GQ Honeyfarm traces.

2.3.6 *Global Address-Misconfigured P2P Traffic Estimation*

Honeynet	LBL	NU			Mean
/8 Prefix	P1	P2	P3	P4	
# of /24	5	6	2	2	
Percentage	32.4%	41.9%	44.2%	37.0%	38.9%

Table 3: Percentage estimations (07/2007 – 12/2007 data of LBL and NU Honeynet datasets).

We have estimated the current percentage of address misconfiguration traffic out of background radiation traffic throughout the whole Internet. We use the recent data (07/2007 – 12/2007) from the LBL and NU Honeynets for the analysis. We treat the percentage estimated from the Honeynet sensors in each /8 prefix as an independent sample when estimating the percentage for the whole Internet. We then use the mean of these percentages from different samples as a more representative result for the whole Internet. The NU Honeynet spans three different /8 prefixes. The LBL Honeynet is in a single /8 prefix. We estimate the percentages for the four /8 prefixes as shown in Table 3. *The average percentage of address-misconfigured P2P traffic in the Internet background radiation is 38.9%*. It is very likely that the percentage will continue to increase.

We also try to estimate the total bandwidth consumed by address-misconfigured P2P traffic throughout the whole Internet. It is very challenging, given the limited data we have. Currently, we use linearly extrapolation as the first order estimation, which might not be very accurate. The traffic speed of the four prefixes pass the Chi-square Goodness of Fit test with p-value=0.9, which suggests that the traffic speed on different prefixes might follow a uniform distribution. We linearly extrapolate the mean value of the traffic speed of the four prefixes. Since 94% of traffic seen in Honeynet is TCP traffic and most peers run Windows, we will use the Windows TCP stack behavior to estimate the bandwidth consumption. We assume that a random host will not reply to the P2P probes. Therefore, the peers only resend three SYN packets of 60 bytes each. If the host replies with RST packet or if the port is open, then we will see more traffic but as a conservative estimation we ignored such cases. Therefore, in each connection we see 180 bytes. Based on this we estimate that 7.90Gps (2^{24} /24 networks in the Internet each having $0.327 \times 180 \times 8$ bits) bandwidth is wasted by address-misconfigured P2P traffic in the whole Internet.

France	58.0%	
China	6.0%	
United States	5.3%	
Spain	4.4%	
Italy	4.3%	

Table 4: The top five countries that contributed most eMule misconfigured peers in the LBL dataset.

China	93.2%	
United States	2.5%	
China - Taiwan	1.3%	
Canada	0.6%	
Japan	0.4%	

Table 5: The top five countries that contributed most BitTorrent misconfigured peers in the LBL dataset.

2.3.7 *Where Do Misconfigured Peers Come From?*

We use IP to location mapping services to find the locations of misconfigured peers.

For eMule, we find the LBL and NU data agree well with each other. Most of the peers which target the Honeynets are located in Europe and from a small number of ASes. Table 4 shows the top five countries where misconfigured peers in LBL data are from. We ignore the NU result since they are similar.

For BitTorrent, both LBL and NU datasets suggest that China and United States is the place where most misconfigured peers from. Table 5 shows the top five countries which contribute most misconfigured peers in the LBL dataset. Further analysis suggests a large portion of US-originated misconfigured peers belongs to anti-P2P companies. The more detailed analysis of anti-P2P related behavior is shown in Section 4.3.1.

This confirmed the prevalence of eMule in Europe and BitTorrent in China [33].

3. SYSTEM DESIGN

3.1 General Design Principles

In general, two reasons can potentially cause P2P misconfigurations, the software bugs and the external attack or misconfiguration injection.

We find two approaches are useful for detecting the misconfiguration: (i) measurement including both passive monitoring and active backtracking; (ii) tracking the information flow for suspicious P2P software in a controlled environment. The first approach helps quickly identify which software version, communication mechanism or peer groups potentially are the sources of the misconfiguration. It is especially useful when there are a large number of software versions or peer groups. This approach also helps identify the possible attacks or misconfiguration injected such as from anti-P2P companies. The second approach can further validate misconfiguration caused by software bugs.

After detection, we need to consider how to diagnose the root causes of the misconfiguration. This diagnosis step is much harder than detection, and usually requires manual inspection. Many distributed system debugging approaches can be used in this step. We find information flow tracking is still very useful in this stage. Another useful approach is to form the hypotheses regarding the root causes and to conduct experiments to validate them.

3.2 Design of P2PScope

As shown in Figure 5, P2PScope has two major subsystems: (i) detection subsystem, (ii) root cause diagnosis subsystem. The detection subsystem is for detecting the address-misconfiguration. The root cause diagnosis subsystem is for further root cause diagnosis.

The detection system has three modules: (i) passive monitoring module, (ii) backtracking module, and (iii) information flow tracking module. The passive monitoring module detects the address-misconfigured P2P traffic from Honeynets. Further, backtracking module collects more information about the misconfigured peers and the file hash related to address misconfiguration. When these two measurement modules detect some suspicious P2P software version, the software will be passed to information flow tracking module. In the information flow tracking module, we install that P2P software version and check its peer propagation. Finally, all the useful information will be output to root cause diagnosis subsystem for further analysis.

3.2.1 Passive Monitoring Module

The common practice to study Internet background radiation is to set up a Honeynet or Honeyfarm on unused IP blocks. The Honeynets or Honeyfarms respond to the incoming connections and generate "faked" responses. For our purpose of P2P address misconfiguration study, we use Honeyd-based [30] low interactive Honeynets. Honeyd is a

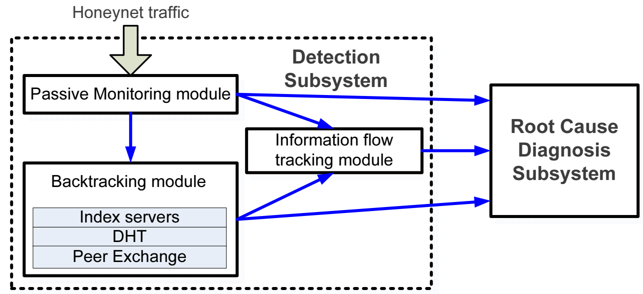

Figure 5: Architecture of the P2PScope system.

flexible lightweight framework for simulating different network protocols. Simulating 10 /24 networks on a Pentium4 machine uses on average less than 5% of CPU and around 40MB of memory. The original Honeyd can only configure the responder based on port numbers. However, P2P traffic mostly targets random port numbers.

To overcome this limitation, we apply a *payload signature based* approach. None of the existing tools support such an option. We modified the Honeyd to achieve this. We first identify the P2P protocol based on payload signatures and then call appropriate P2P responders for further processing. Currently, we have developed BitTorrent and eMule protocol responders. Although the P2P protocol signature might have false positives, the protocol responders will further parse the protocol messages accurately and the false positives will be discarded. For unknown protocols, we use an "echo" responder and hope it can trigger some more detailed protocol messages. Indeed, it is quite useful, given that many protocols are symmetric.

For BitTorrent and eMule, when we identify the protocol of a connection, we send the corresponding BitTorrent/eMule handshaking messages and also exchange peers from the remote host if possible.

3.2.2 Backtracking Module

To understand how the peers get misconfigured, the backtracking system tracks the peers in real-time. There are several challenges for designing and implementing the backtracking subsystems. First, current file sharing systems, such as BitTorrrent and eMule, use multiple ways to map a file hash (the unique identifier for a file) with its sources, such as index servers (trackers in BitTorrent terminology), DHT (Distributed Hash Table) and peer exchange protocols. Usually all these methods work together, which make the diagnosis harder. Second, some file sharing systems such as BitTorrent involve multiple incompatible implementations for certain protocols. For example, BitTorrent has two DHT implementations, the Mainline Kademlia DHT and the Azureus Kademlia DHT. Moreover, different BitTorrent implementations such as Azureus, uTorrent, and BitComet, use different peer exchange protocols. Finally, some P2P software and protocols are either a closed-source or lack documentation. In our system we try our best to implement modules for each of these protocols.

Index Server Querying: For eMule, we gather 185 index servers by searching popular "server.met" files using popular search engines and merge them. For BitTorrent, we pre-compile a list of top 100 trackers (index servers) by (i) extracting the tracker information from top 10,000 torrent files downloaded from the Internet, and (ii) augmenting the list from various tracker ranking sites and BitTorrent forums. We implement light weight eMule and Bittorrent clients to query multiple servers simultaneously.

Peer Exchange Querying: Once a peer knows some sources of a file object, it can query them to get more sources. Most of the popular file sharing systems such as eMule and Bittorrent support this functionality. It works like an unstructured P2P network. For eMule, we have implemented the eMule source exchange protocol. For BitTorrent, since it has multiple incompatible versions of peer exchange protocols, we first check the software version and then use corresponding protocols.

DHT Querying: The DHT crawler will track all the peers which can be obtained from querying DHT for a given file hash. There are two popular parallel implementations of BitTorrent DHT protocol: Azureus and BitTorrent Mainline. We have developed DHT peer modules for both implementations.

In the backtracking system, for a given file hash, we periodically queried the index servers or DHT for the peer lists every three minutes. We have tried a shorter period, but find the server tend to give the same peer list as that in the previous period. Then, for any peer, we used peer exchange protocols to query more peers at once. The whole process will stop when the total number of peers remain stable for 30 minutes. We use the same number of threads for server queries and peer exchange queries.

3.2.3 *Information Flow Tracking Module*

Whenever a certain P2P client version is suspicious, we install it in our controlled monitoring environment. In the controlled minitoring environment, by interpreting the protocols (index server, DHT, and peer exchange protocol) used for peer propagation, we monitor the peers in and out. If the software gives out a peer which never comes in, we know the software will cause address misconfiguration.

3.2.4 *Root Cause Diagnosis Subsystem*

Tracking down the root cause of address misconfiguration is a challenging problem. we have developed the following tools for understanding the root causes: (i) Track the information flow within the suspicious P2P software. (i) Trace the Honeynet IP addresses propagated in the P2P systems. (ii) Check for the routability of the peers returned. (iii) Check whether the peers are on the anti-P2P IP lists and analyze their behavior. (iv) Check the reverse byte order of peer IPs because it is one of the most frequent mistakes in networked system implementations.

Most of the large events are caused by BitTorrent and eDonkey/eMule. Thus, our current diagnosis focuses on these two P2P protocols. Still, P2PScope can be easily adopted to other P2P protocols/software.

To understand why the bogus peers can be produced by the suspicious P2P software, we can apply the taint check techniques, to trace how the bogus peers are produced. we can also apply existing distributed system debugging techniques in the controlled environment. For instance, we can change the software to add logging functions, and let the program output the internal states of the program for further diagnosis.

Another toolkit that helps the diagnosis is the routablity checking. In [14], Cooke *et al.* find that 66.8% of all possible IPv4 addresses are unroutable. If we assume that the bogus peers are produced randomly, then we would find a reasonably large percent of unroutable bogus peers. If the percentage is small, then it implies that bogus peers are not produced or injected randomly. We use the IP to AS mapping services of Routeviews with 50+ BGP feeds to check routability. The percentage of unroutable peers can serve as a low bound of the bogus peers.

Since we suspect the anti-P2P companies, which try to take down the P2P networks, might be related to address misconfiguration, we also check whether the peers belong to anti-P2P IP blacklist. We compile the anti-P2P IP blacklist from several resources, (i) the anti-P2P blacklist software Peer Guardian [6], (ii) various P2P discussion forums, and (iii) the list published in [11]. The list might not be fully comprehensive but covers all the famous anti-P2P companies we know.

With the above tools, we mainly try to answer the following questions: (i) which peers spread misconfiguration? (ii) what is the root cause? (iii) how is misconfiguration disseminated? (iv) what is the percentage of invalid peers in misconfigured P2P networks? (v) how badly are individual clients affected by misconfigurations?

4. DIAGNOSIS AND ANALYSIS

We have deployed P2PScope in the NU Honeynet for realtime monitoring and diagnosis since Aug. 2007. The measurement and diagnosis results are pretty consistent. We mainly use one-month data for illustration purpose in this section.

We first study the general characteristics that are related to the root causes of all the P2P systems. Then we focus on two case studies for eMule and BitTorrent, which generate most of the misconfiguration traffic.

We classify peers into three major categories: (i) *Misconfigured Peers*—a set of peers that are passively observed by Honeynet; (ii) *Backtracked Peers*—a set of peers actively observed through backtracking; (iii) *Reverse Honeynet Peers*—a set of peers obtained by reversing the target IP of the Honeynets.

4.1 Data Plane Traffic Radiation

Usually, there are a number of different protocol messages involved in a P2P file sharing network. Address misconfiguration in all the six P2P systems that we explored has a common feature: the traffic is caused by file downloading requests, *i.e.*, data plane traffic. Different P2P systems use different ways of communication for exchanging control plane information (*e.g.*, Index Server, Peer Exchange Protocol and DHT), but all have the same problem of giving bogus IP addresses to peers. These bogus IP addresses propagated quickly in P2P systems and caused large amount of address-misconfigured P2P traffic.

4.2 eMule Misconfiguration Diagnosis

We find that one of the major reasons of address misconfigurations in eMule is a byte ordering problem in certain implementations of eMule source exchange protocol.

4.2.1 Which Peers Spread Misconfiguration?

Hypothetically, the misconfigured peers that target the Honeynets might include anti-P2P peers, the peers from anti-P2P companies that are hired to prevent Internet piracy. However, we did not find any event with a high percentage of anti-P2P peers. From an analysis of the historical events of LBL and NU Honeynets, we have found that at least 99.24% of the misconfigured peers in any event are normal peers. The total number of misconfigured peers observed from the Honeynets are 128,877. For real-time backtracking, we obtained a similar percentage. Therefore, *we conclude that normal peers are mainly involved in address misconfigurations in eMule.*

In addition, the files involved in an eMule event is quite diverse. The median of number of files involved in an event is 28. This indicates that the address misconfiguration in eMule is not caused by specific files shared in the eMule network. Furthermore, at least 40% of files in our datasets should be of no interest to the anti-P2P companies. Those 40% files are documents, software, games and some very old videos (at least one year before the event). On the other hand, anti-P2P companies are usually only interested in latest movies/TV shows/musics, because these files are more related to the movie/music industry's revenue. Therefore, we believe that these events are unlikely to be caused by anti-P2P companies.

4.2.2 What Is the Root Cause?

The byte ordering problem hypothesis. When we analyze the network level behavior of the eMule misconfiguration, we find that the IP addresses formed by reversing the byte order of the targets were mostly alive. We tested 10 events within the same day as they occurred; 8 of such reverse IP addresses were indeed alive. This led us to a hypothesis that the eMule misconfiguration is caused by a network byte ordering problem.

Validation of the byte order problem hypothesis. With the real-time P2PScope system, we have done two experiments to verify that it is indeed a byte ordering problem, and finally we have confirmed the hypothesis with source code study.

The first experiment is to check whether the reverse target IP is running eMule on the same port number as the target IP. Since eMule usually uses a random port number, the chance of false positives for the reverse IP running eMule with the same port number is very small. We have tested 13 targeted destination in real-time, and have found 61% of them are indeed running eMule on the same port number. This strengthened our belief that it is mainly a byte ordering problem. For the remaining 39% of cases in which eMule is not running on their reverse target IPs, we conjectured the following: due to the dynamic nature of the P2P network, even when we crawl peers in real-time, there are still many peers with which we cannot connect (Figure 6). Still, the byte ordering problem appears to be the major cause.

In the second experiment, after retrieving a file hash found in real-time, we crawl the set of backtracked peers that had the file hash, and checked the percentage of IP addresses that are unroutable. Obviously, those IP addresses are caused by misconfigurations. Then, we checked (among the unroutable IP addresses) for the percentage of IP addresses whose reverse IP addresses are running eMule. We aggregated the result of 100 file hashes found in real-time. As shown in Figure 6, we find 10.3% of backtracked peer IPs are unroutable. Among them, 69.6% had a reverse IP that runs eMule. This, again is another strong evidence in favor of the byte ordering problem hypothesis.

We further find all aMule versions before 2.1.0 indeed have the byte order problem. AMule is an popular eMule alternative which can run on different platforms including Windows, Unix and Mac, while official eMule only supports Windows. EMule has a complex client ID to IP mapping system called HybridID. For simplicity, here we only discuss High ID peers (peers with public Internet IPs). For High ID peers, if a peer is from index servers, its client ID is its IP in network byte order. On the other hand, if the peer is from KAD (eMule Kademlia DHT), the client ID is its IP in little-endian. Therefore, when packing source exchange messages, the client has to check where the peer is from and then change the byte order when necessary. However, in aMule with versions before 2.1.0, this problem has been ignored. AMule assumes peers are from index servers, but actually some peers are from KAD. This bug has been fixed in aMule 2.1.0 version according to their Changelog file. However, some users still use obsolete versions. Due to the complexity of the HybridID system, it is highly likely that other eMule alternatives have the similar bugs.

4.2.3 How Is the Misconfiguration Disseminated?

Given a file hash, there are three ways to get the corresponding peer list through eMule: index servers, eMule source exchange protocol and eMule Kademlia DHT. For most eMule variants, people use all three simultaneously. The question is which one is responsible for the misconfiguration dissemination.

DHT: We have not observed any eMule Kademlia DHT packets towards the LBL and NU Honeynets at all. Therefore, our conclusion is that the eMule Kademlia DHT protocol does not have the byte ordering problem.

Index servers or source exchange protocol: Using the approach described in Section 3.2.2, we have queried 100 file hashes and got a total of 37,079 unique peers. Table 6 shows the comparison of different characteristics of the peers re-

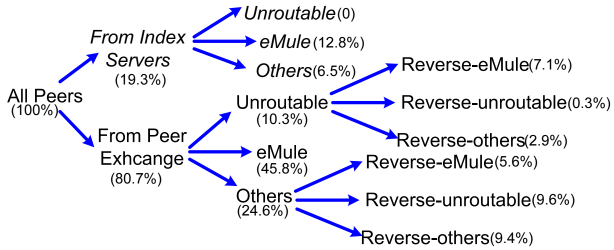

Figure 6: The peer distribution of eMule from active backtracking. "Unroutable" refers to those peers that are unroutable; "eMule" refers to those peers that are running eMule which is verified by probing into them with eMule protocol; and "others" are all the remaining routable peers. "Reverse" means we study its reverse IP.

turned by index servers and the peers from source exchange protocol.

Characteristics	From index servers	From source exchange
% in total	19.3%	80.7%
Returning Honeynet IP	0	10.7%
Unroutable	0	12.8%
Whose reverse IPs run eMule	0	15.7%

Table 6: The characteristics comparison of peers returned by index servers and source exchange.

We find that when 3,208 (10.7% of the peers returned by source exchange protocol) of peers are queried for source exchange, they gave the Honeynet IP addresses. But, the index servers never returned any Honeynet IP. We have also done a routability check. 12.8% of peers returned by source exchange protocol are unroutable. On the other hand, none of the peers returned by index servers are unroutable. Also, there are 15.7% of peers from source exchange whose reverse IPs run eMule. None of peer returned by index servers has such behavior.

Therefore, *it is not the index server but the source exchange which was used for disseminating the misconfiguration*.

4.2.4 What Is the Percentage of Bogus Peers in the Misconfigured P2P Networks?

We try to estimate the percentage of peers (from the source exchange protocol) which have the byte ordering problem. This is challenging because there are certain cases in which we cannot determine whether or not they had a byte ordering problem. Instead, we estimate its lower and upper bound and make the bounds as tight as possible.

We estimate these bounds based on the distribution of peers from the peer exchange shown in Figure 6. If a peer is not running eMule but its reverse IP is, we believe it to definitely have a byte ordering problem. Such peers comprise 12.7% of total peers (*Unroutable→Reverse-eMule* and *Others→Reverse-eMule*).

if a peer is unroutable, but its reverse IP is not, we believe it is likely to have a byte ordering problem. 10.0% of total peers have such a property. We know *Others→Reverse-eMule* peers are caused by byte ordering problem, and the *Others→Reverse-others* is possibly caused by byte ordering problem. Therefore, the upper bound is $10.0\% + 5.6\% + 9.4\% = 25.0\%$ of peers.

4.2.5 How Badly Are Individual Clients Affected by Misconfigurations?

To calculate the percentage of infection in the peer list of each client, we have actively tracked all the misconfigured clients connecting to NU Honeynet. We connect to clients in real-time and retrieved their peer list through peer-exchange messages. We calculate the mean percentage of unroutable IP addresses in the peer list of all the incoming clients. This served as an average lower bound of infection. We then try to connect to all clients' peers to checked whether they are running eMule or not. The mean percentage of IPs to which we cannot connect are considered an average upper bound of the infection. We have checked 4,509 misconfigured peers. The average bounds of infection in the peer list of each client are [7.4%, 22.8%]. These percentage bounds are lower than the bogus peer percentage we estimated before. This is because after several retries, the bogus peers are more likely to be removed from the peer list than normal peer IPs.

Among all the peers, 6.3% peers are heavily influenced by misconfiguration. For these clients, more than 80% of peers in their peer list are unroutable. The rest of the clients have a moderate level of infection. We have not found any easy way to differentiate these peers except routability checking.

4.3 BitTorrent Misconfiguration Diagnosis

4.3.1 Which Peers Spread Misconfiguration?

When analyzing the historical events, we have found that BitTorrent events displayed two exclusive distinct sets of network-level characteristics. For example, in some events each source IP sends out a large number of connection requests (> 100) during its occurrence, while the sources of the other events had a much smaller request rate.

For events with large request rate per source IP, we find that the incoming sources were mainly from a small number of /24 networks. The analysis of NU data is shown in Table 8. We use the concept "*visit*" to measure the popularity of source /24 networks. An IP address appearing in an event counts as one visit from the corresponding /24 network. If IP appears in two different events, we count it as two visits. Table 8 shows the /24 networks with more than one visit. The remaining 120 /24 networks contribute only one visit per /24 network. These /24 networks with more than one visit are from a small number of server collocation companies.

Except for one Verizon /24 network, all other networks belong to anti-P2P companies Media Defender [4], MediaSentry corp. [5], Net Sentry *etc.*. XEEX is a network provider that leases IP ranges to other companies. Some of the XEEX IP range is owned by the *Media Defender* [4], a company that is hired by copyright owners to log IP addresses of peers. This company owns a back end of 2,000 servers collocated around the world. The discussion groups

	Normal peer events	*Anti-P2P peer events*
# of events	136 - NU, 36 - LBL	75 - NU, 39 - LBL
Client Software Used - NU	90% - UTorrent 10% -Others	100% - Azureus
Client Software Used - LBL	57% - Bit Spirit, 31% - BitComet 12% -Others	100% - Azureus
Message Length	Variable lengths	Similar lengths
Files Queried	Diverse	Latest music and movies
Average No. of Connections /IP	25	400
Source Correlation	Little coordinated	Highly coordinated
Arrival & Departure	Gradually	All together
Avg Event Duration	106.1 hours	4.5 hours
IP Distribution	Diverse	Small number of /24

Table 7: Comparison between normal peers events and anti-P2P peer events.

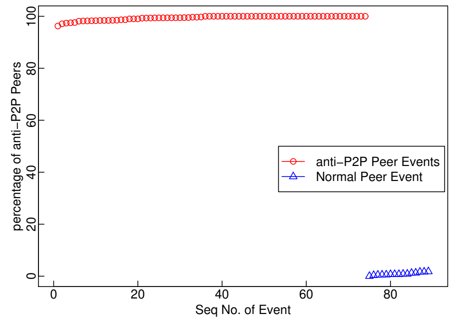

Figure 7: Percentage distribution of anti-P2P peers in NU BitTorrent events.

hosted on the Peer Guardian [6] website mention XEEX frequently as blacklisted by users.

#visits	/24 networks	AS	Nation	Company	Usage
2304	66.117.5/24	27524	US	XEEX	Server Farm
1461	66.184.194/24	27524	US	XEEX	Server Farm
956	72.35.224/24	36607	US	fuzioncolo	Server Farm
647	66.172.69/24	27524	US	XEEX	Server Farm
346	63.216.76/24	27524	US	XEEX	Server Farm
205	216.151.155/24	27524	US	XEEX	Server Farm
192	63.219.21/24	27524	US	XEEX	Server Farm
166	66.180.205/24	11051	US	Cyberverse	Server Farm
21	154.37.66/24	2914	US	cogentco	Server Farm
11	65.120.42/24	209	US	VERIZON	Unknown

Table 8: The popularity of the source /24 networks observed in anti-P2P peer events.

We use the anti-P2P IP blacklist to classify the sources in each event and calculate the ratio of anti-P2P peers to the total P2P peers. All events are at one of the two extremes; an event mostly consists of either normal peers or anti-P2P peers. No event is a mixture of both. The events dominated by anti-P2P peers surprise us, since they represent the cases that *the anti-P2P company themselves somehow get misconfigured solely*.

Figure 7 shows the percentage of anti-P2P peers in each event from the NU Honeynet. The x-axis is event ID. The percentage of anti-P2P peers are in either extreme. The results from LBL Honeynet are similar. We call anti-P2P peer dominated events *anti-P2P peer events* and normal peer dominated *normal peer events*. We observed anti-P2P peer events in different locations (in both LBL and NU datasets). This suggests the prevalent nature of misconfigured anti-P2P systems. Moreover, we have seen most (6 out of 7) of the well known anti-P2P companies in our Honeynet. There might be more anti-P2P companies we are not aware of.

We compare the properties of these two types of events and summarize the results in Table 7.We find peers from anti-P2P peer events all use Azureus clients. We conjecture anti-P2P companies modify Azureus clients and add their functionalities, because Azureus is a popular open-source BitTorrent client with a nice plug-in architecture. On the other hand, peers from normal peer events runs on a diverse set of clients. The peers in anti-P2P events are highly coordinated. They start to contact the honeynet IPs in almost the same time. They also depart in the same time. On the other hand, the peers in normal events arrive and depart gradually. The messages sent in anti-P2P peer events are almost same. They have similar message length and even the similar bitmap that annotates the content availability distribution. To some extent, the group of anti-P2P peers are like a clone army with identical soldiers. Moreover, the peers in anti-P2P peer events are from a small number of networks. The metric source correlation measures the degree to which the sources of different events are overlapped. 97% of pairs of anti-P2P peer events share more than 25% of sources. On the other side, the normal peer events are little coordinated.

4.3.2 *What Is the Root Cause?*

There are numerous compatible BitTorrent clients, written in a variety of programming languages, and running on a variety of computing platforms. This makes the root cause diagnosis a very challenging job. We have not captured any real-time anti-P2P events. Thus, we cannot diagnose their root causes. In this section and later, we mainly focus on the normal peer events. Normal peer events can still be caused by anti-P2P peers as we discuss the next.

Root Cause I: anti-P2P companies deliberately inject the bogus peers in the P2P system. Our P2PScope system tracks the misconfigured peers in real-time. Out of the 9834 misconfigured IP addresses monitored, only 52 hosts' peer lists have any unroutable IPs. Of these, nine have more than 50% unroutable IPs in their peer list, and 43 hosts have less than 3.2%. Surprisingly, all of these nine hosts come from the same server farm 72.172.90/24, which is an IP block owned by an anti-P2P company called Artistdirect. Through further analysis on the server farm, we find each of these IPs runs 100's of BitTorrent clients simultaneously. The IPs also support VMWare authentication protocol. We conjecture the anti-P2P uses VMware based virtual machines to run multiple BitTorrent clients. All of the clients run a version of Azureus 2500 which supports all the existing protocols (including the uTorrent PEX protocol), while the vanilla Azureus clients support only a subset of the protocols. Moreover, unlike vanilla Azureus clients, these anti-P2P clients respond to any arbitrary file hash, *i.e.*, declare they have any file the remote peer wants and exchange the bogus peers with it. We conjecture Artistdirect have modified the Azureus client for their purpose.

We have checked the peer lists returned by these nine anti-P2P IPs. They returned all bogus peers. Among the peers, 56.7% are unroutable, 1.8% of them are from the same server-farm, and we could not connect to any of the remaining 41.5% of the peers. Only 4% of the IPs responded to our ping requests. We believe these anti-P2P peers randomly choose bogus peers. We suspect anti-P2P companies want to slow down the downloading process by supplying bogus peers.

For the normal-P2P peers, we further classify them into two groups based on whether their infohash (file hash in BitTorrent terminology) is ever related to an anti-P2P group (*i.e.*, whether some peers which query the same infohash have anti-P2P peers in their peer list). The first group of infohash and its corresponding normal-P2P peers are not related to anti-P2P peers. The second group of normal-P2P peers are related to anti-P2P peers. For the second group, most of the peers ($> 90\%$) are related to anti-p2p peers within two peer hops.

The anti-P2P related normal peers are 19.7% of the total misconfigured peers and are responsible for 20.7% of the probes send to the Honeynet. *Therefore, the anti-P2P companies are responsible for 1/5 of the traffic observed.* We mainly identify anti-P2P peers by known anti-P2P blacklist. This estimation is quite conservative. The real influence caused by anti-P2P companies could be larger than what we report here.

Root Cause II: the byte order problem of KTorrent clients. We further backtrack the misconfigured infohashs (BitTorrent file hash), which we extract from messages of the misconfigured peers. We collected 9,000 unique peers in total. Among them, 13 peers are unroutable. We find most of the unroutable peers can be obtained by peer exchanging with KTorrent clients. Therefore, we believe KTorrent is suspicious. We also study the UTorrent and Azureus given that they are the most popular BitTorrent clients.

We setup a controlled BitTorrent farm on three machines and run popular clients on them as discussed in Section 3.2.3. We run three clients, each for one software type (*UTorrent, Azureus, and KTorrent*) with default configurations. The captured traffic is processed later to debug the system. We download top 7 torrent files seen in Honeynet from the Internet using the clients.

Number of IPs	*AZUREUS*	*UTORRENT*	*KTORRENT*
Total	9226	41265	5933
Outgoing	1067	8921	1161
Outgoing - Incoming	0	0	849
Unroutable	152 (1.6%)	1051(2.57%)	753(12.6%)
(Outgoing - Incoming) and Unroutable	0	0	478(56%)

Table 9: PEX IP propagation statistics of different Bittorrent client.

We categorize the peers seen by our clients into two parts: (i) incoming peers—peers that comes to the installed clients through trackers (index servers in BitTorrent terminology), PEX messages, DHT, and the sources of incoming connections. (ii) outgoing peers—peers that are propagated by the clients through PEX messages and DHT. In Table 9, we have shown the statistics of different BitTorrent clients run in our controlled experiments. We analyze *outgoing - incoming* peers that leave our system. Those peers are the bogus peers created by the client itself. These outgoing bogus peers are the main cause of address misconfiguration. The client that gives out bogus outgoing peers is the culprit. As shown in the Table 9, among all outgoing KTorrent peers, 56% of them are bogus. Thus, it shows that KTorrent is the dominating source of spreading misconfigurations. KTorrent clients give out bogus IP in the system using the Peer Exchange(PEX) messages. Similar to the eMule case, we found most of all the unroutable IPs, after reversing the byte order, indeed run BitTorrent.

```
void UTPex::encode(...)  // UTPEX.CPP
{
        .
        .
        // Use KDE API to get an IP in Network Byte Order.
        // WriteUint32 swap the byte order again!!!
        WriteUint32(buf,size,addr.ipAddress().IPv4Addr());
        .
        .
}
void WriteUint32(Uint8* buf,Uint32 off,Uint32 val)  //FUNCTIONS.CPP
{
        // swap the byte order
        buf[off + 0] = (Uint8) ((val & 0xFF000000) >> 24);
        buf[off + 1] = (Uint8) ((val & 0x00FF0000) >> 16);
        buf[off + 2] = (Uint8) ((val & 0x0000FF00) >> 8);
        buf[off + 3] = (Uint8) (val & 0x000000FF);
}
```

Figure 8: Code Snippet of KTorrent Byte Order Bug

Then we studied the source code of KTorrent to pinpoint the exact bug. The problem is *Reverse Byte Order Problem*. In Figure 8, we show the code snippet related to this bug. The UTorrent PEX protocol requires the IP addresses in the peer exchange messages to be network byte order. When KTorrent sends the peer exchange messages out, in its *encode* function, it reverses the byte order twice, so that finally IP addresses become host byte order again. *addr.ipAddress().IPv4Addr()* is a KDE library method which return IP addresses in network byte order. However, in *WriteUint32* function implemented by KTorrent, it reverses the byte order again; thus cause the problem. Since this bug happens to the outgoing messages that do not directly reduce the KTorrent client performance, it might be less obvious to the developers. To our knowledge, we did not find anyone else reports this bug before.

4.3.3 How Is the Misconfiguration Disseminated?

BitTorrent DHT: Along with the normal BitTorrent traffic, we have also observed some Mainline BitTorrent DHT traffic in our Honeynets. It is about 5% of BitTorrent traffic. We did not observe Azureus DHT traffic in the Honeynets. We studied whether the Mainline DHT could disseminate the misconfiguration. We gathered a list of 3345 unique BitTorrent peers by querying the DHT in real-time. We then tried to connect to every client to check whether they are anti-P2P, Honeynet or unroutable peers. We found that none were Honeynet, anti-P2P, or unroutable IPs. Based on this, we conclude that *it is unlikely that Mainline DHT is respon-*

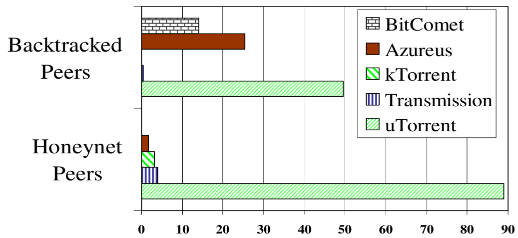

Figure 9: Client software distribution comparison (percentage)

sible for spreading or generating misconfiguration.

Tracker or Peer Exchange (PEX): Each major BitTorrent implementation supports an incompatible version of peer exchange protocols. Major groups are: uTorrent, Azureus and BitComet. UTorrent PEX protocol is the most popular one. Most other clients including Azureus support this protocol. Azureus and BitComet also have their own peer exchange protocol. The question we try to answer is: Is the misconfiguration propagated through the PEX protocols? And if so, through which one?

We have conducted two experiments below and have found that *the misconfiguration is mainly spread through the uTorrent PEX protocol, but not trackers or other PEX protocols*.

First, we have tested 20 targeted destinations (Honeynet IPs) obtained in real-time. We cannot find any of them by querying trackers. In contrast, for 15 out of 20, we obtain the Honeynet IP through backtracking other peers via uTorrent compatible PEX protocol.

In the second experiment, we study the peer exchange protocol distribution in the misconfiguration traffic. Any peer can only exchange peer information in its own group. Bugs or intentional poisoning in a particular peer exchange protocol implementation can only spread misconfiguration within that group. Using the backtracking methodology described in Section 3.2.2, we gathered 9,000 backtracked peers in total. We have compared the 2794 incoming Honeynet peers (*i.e.*, misconfigured peers) with this backtracked peer list. We find that more than 80% of Honeynet peers are on this list, though not directly obtained from trackers. Figure 9 shows the percentage of the top five client programs of the Honeynet peers vs. those of the backtracked peers (*i.e.*, any peers sharing the same info hash). If all the PEX protocols are used for propagation, the two distributions should be similar. However, they are significantly different. Honeynet peers are predominantly clients that support the uTorrent PEX protocol.

To understand how the bogus peers propagate through the uTorrent PEX protocol, we study the behavior of different uTorrent PEX compatible clients. We classify the behavior into three categories: (i) IP with port closed, (ii) IP responds with SYN-ACK (only active port and not running Bittorrent), (iii) IP running Bittorrent and responds with BT-HANDSHAKE messages. In Table 10, we show the results. KTorrent propagates anything that comes to it without even checking for an open port. UTorrent propagates anything which has an open port. Azureus only propagates peers with active BitTorrent client. Therefore, KTorrent will fully propagate the bogus peers. UTorrent will propagate the bogus peers if they happen to have the port open. Azureus usually will not propagate any bogus peers. To reduce the bogus peers, we believe all the clients should have strict checking like Azureus. In this way, the address-misconfigured P2P traffic can be greatly reduced.

	NO CHECK	*SYN-ACK*	*HANDSHAKE*
KTorrent	Yes	Yes	Yes
UTorrent	No	Yes	Yes
AZureus	No	No	Yes

Table 10: Peer IP propagation criteria of different clients. "Yes" means the IPs will be propagated under that condition

4.3.4 What Is the Percentage of Bogus Peers in the Misconfigured P2P Networks?

We realize that estimating the percentage of bogus peers is very difficult, so we try to estimate the lower and upper bounds of this percentage instead. The P2PScope system collect the peer list from both trackers and peer exchange for each of the files. For 30 files, in total, we have collected 9,000 IP addresses of peers. Among them 13 IP addresses are unroutable. Those 13 IP addresses are associated with 2 files. The percentage is 0.14%.

We also estimate the upper bound of the bogus IP percentage by the percentage of all the peers we cannot connect to, which is about 35%. Therefore *the percentage bound of bogus peers is [0.14%, 35%].*

4.3.5 How Badly Are Individual Clients Affected by Misconfigurations?

We have actively tracked the misconfigured traffic of the normal peer events coming to NU Honeynet. We have collected 9,834 unique hosts. We connect to these peers in real-time and got their peer list through peer-exchange messages. We have calculated the mean percentage of unroutable IPs in the peer list of all the incoming clients. This serves as a lower bound of infection (bogus peers). We then try to connect to all the peers from the list to check whether they are running BitTorrent or not. The mean percentage of IPs to which we are unable to connect is considered the upper bound of the infection. Most misconfigured peers do not have a high percentage of bogus peers in their peer list. *The average percentage bound of infection (bogus peers) in the peer list of each client is [0.053%, 32%].*

4.4 Validation with External Misconfiguration Events

Most of our study is based on the LBL and NU Honeynet sensors. One question is how representative these results are and whether they have any bias. To answer this question, we have examined potential external address-misconfiguration events and checked their properties.

We randomly select 15 popular files (popular movies and some popular Linux software) and search for all related peers as discussed in Section 4.2.3 for eMule and Section 4.3.3 for BitTorrent. We then checked if any of these peers were unroutable, and if so, we took these peers and the related file download as misconfiguration events and analyzed their properties. Of course, there were many properties for which we did not have sufficient information such as those observed in the Honeynet. We did, however, find that the statistics are pretty similar to those misconfiguration events observed in our Honeynets. As an example, let us consider the bounds of bogus peer percentage.

We apply exactly the same methodology as presented in Sections 4.2.4 and 4.3.4. For eMule, we found a total of 10,267 peers collected, and the percentage bounds for bogus peers to be [8.6%, 24.4%], which is very similar to [12.7%, 25.0%] bounds calculated with our Honeynet events (see Section 4.2.4). Similarly, for BitTorrent, we found 3 unroutable IPs out of a total 20,690 peers collected. The percentage bounds for bogus peers were [0.00015%,34%], very close to the bounds derived from our Honeynet events, [0.14%,35%] (see Section 4.3.4). Therefore, *we believe our findings are representative of the address misconfigurations on the Internet*.

One important aspect of our study is to compare and contrast the misconfigured P2P traffic (in particular, by normal peers) vs. the normal P2P traffic. The question we want to answer is whether we can filter the misconfigured traffic by analyzing the network level characteristics. Our finding is that the characteristics look almost identical. Hence, filtering based on network-level characteristics is not applicable.

4.5 Implications and Possible Solutions

Build the anti-P2P IP list: Through our study, we find several unique characteristics of the anti-P2P peers. Although we do not want to take sides in the P2P war, those characteristics can be used to detect future anti-P2P IPs, given the anti-P2P companies usually change their IP range frequently. The following criteria can be used for detection: (i) peers that only return non-functioning peers through peer exchange protocol, (i) peers which declare that they have any file you want, (iii) peers that repeatedly contact you again and again (*e.g.*, more than 100 times).

We validate this detection mechanism using existing anti-P2P blacklist we collect, and do not observe any false positives. We also do detect anti-P2P peers that are not in the anti-P2P blacklist we have. Thus, we believe it is useful to be used for detecting future anti-P2P IPs.

Detect bugs in the P2P software: We find the information flow tracking techniques are useful for confirming the bugs in the P2P software. We have reported the KTorrent bug to the developers and they have admitted that bug and promised to fix the bug in their next release.

Suggestions to P2P developers: From analyzing both BitTorrent and eMule IP byte-order bugs, we find that different parts of P2P software have different quality of code. For the parts which heavily influence the peer's own performance, the code quality is high. However, for the parts that is not directly related the performance of the peer itself, such as the sending of outgoing information (*e.g.*, the peer exchange messages), it is much easier to have problems. A general suggestion is that P2P software developers should pay more attention such as test and debugging, on the outgoing information related code.

5. RELATED WORK

Misconfiguration studies: Misconfiguration is widely spread across different network systems on the Internet. Labovitz *et al.* studied wide area backbone failures and concluded that misconfiguration could be responsible for 12% of the incidents [22]. There has also been much recent work that considers the various instabilities and misconfiguration in BGP [27]. A study of firewall misconfiguration [16] by Cuppens *et al.* shows that the existence of errors is likely to degrade the network security policy. However, we find little literature on the study of P2P address misconfiguration.

P2P measurement: The most existing measurement studies of peer-to-peer networks are based on the measurement of the $normal$ P2P traffic [20, 31], while we measure the $abnormal$ P2P traffic observed in Honeynets. This is also being observed by Yegneswaran *et al.* in [34]. Their work focuses on the comparison of network level characteristics of the P2P address misconfiguration with Botnets and Worms, and places an emphasis on accurately classifying Botnet sweeps and worm outbreaks. We mainly study the prevalence and trend of address misconfiguration traffic, and its root causes. In [23], it is found that pollution (both intentional and unintentional) is widespread for popular files. In [13], two other pollution designs are proposed. In [18], the authors develop a deterministic model for pollution evolution in a file-sharing system. All of the above studies focus on $content$ pollution while our focus is $index$ misconfiguration. Liang *et al.* show that P2P systems like Fasttrack and Overnet are vulnerable to index poisoning attacks [24]. In contrast, we investigate the root causes for $both$ intentional and unintentional index misconfiguration.

Distributed System Debugging: Distributed systems are extremely hard to debug. People have proposed various ways to debug distributed systems including replay-based predicate checking [19, 25], online monitoring [26, 32], log forensic analysis [9, 12], network trace based inference [10], model checking [21]. The way we analyze the root causes of P2P address misconfiguration can be viewed as a process of P2P system debugging. However, the following properties make it hard to leverage on any of the above existing techniques. First, the P2P systems we diagnosed are heterogenous. Multiple different implementations exist for a given P2P protocol, such as BitTorrent. Therefore it is hard to apply model checking techniques. Second, the peers are out of our control. We cannot modify the peers to add the logging functionality as required by replay-based predicate checking, online monitoring and log forensic analysis. It is also very hard to log the network traces from remote peers. Third, the P2P systems we monitor are very large consisting of millions of peers. All the existing techniques cannot handle this scale yet. Therefore, we have to come up with light weight approach to diagnose the address misconfiguration.

6. CONCLUSIONS

In this paper, we report the prevalence of address misconfiguration. We find the following: (i) P2P misconfiguration happens in six different P2P systems; (ii) it spreads over the entire IPv4 space; (iii) the address-misconfigured P2P traffic are 38.9% of the Internet background radiation and it is growing quickly.

To detect such misconfiguration and understand the root causes, we have designed the P2PScope system, which combine passive monitoring with active backtracking, and apply information flow tracking techniques.

Using this system, we find that the data plane traffic radiation is the common characteristic of the six P2P systems. For eMule, we found that the problem is mainly caused by a byte ordering problem. For Bittorrent, we found two reasons of the misconfiguration: (i) anti-P2P companies deliberately injecting invalid peers; (ii) KTorrent byte ordering bug in its uTorrent PEX protocol implementation.

7. REFERENCES

[1] Application Layer Packet Classifier for Linux. `http://l7-filter.sourceforge.net`.

[2] Global bandwidth research service executive summary. `http://www.telegeography.com/products/gb/pdf/Executive_Summary.pdf`.

[3] List of bittorrent clients. `http://en.wikipedia.org/wiki/List_of_BitTorrent_clients`.

[4] Media defender, peer-2-peer anti-piracy solutions. `http://www.mediadefender.com/`.

[5] Mediasentry, protecting and enabling online distribution. `http://www.mediasentry.com/`.

[6] Peer Guardian, an IP blocker for p2p system. `http://phoenixlabs.org/pg2/`.

[7] Peer-to-peer networks co-opted for DOS attacks. `http://www.securityfocus.com/news/11466`.

[8] Symantec internet security threat report. Tech. rep., April 2007. `http://www.symantec.com/enterprise/theme.jsp?themeid=threatreport`.

[9] AGUILERA, M. K., ET AL. Performance debugging for distributed systems of black boxes. In *ACM SOSP* (2003).

[10] BAHL, P., CHANDRA, R., GREENBERG, A., KANDULA, S., MALTZ, D. A., AND ZHANG, M. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *ACM SIGCOMM* (2007).

[11] BANERJEE, A., ET AL. The P2P war: Someone is monitoring your activities! In *IFIP NETWORKING* (2007).

[12] BARHAM, P., ET AL. Using magpie for request extraction and workload modelling. In *OSDI* (2004).

[13] CHRISTIN, N., ET AL. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proc. of ACM Electronic Commerce* (2005).

[14] COOKE, E., ET AL. The dark oracle: Perspective-aware unused and unreachable address discovery. In *Proc. of USENIX NSDI* (2006).

[15] CUI, W., ET AL. GQ: Realizing a system to catch worms in a quarter million places. Tech. Rep. TR-06-004, ICSI, 2006.

[16] CUPPENS, F., ET AL. Detection and removal of firewall misconfiguration. In *International Conference on Communication, Network and Information Security* (2005).

[17] DREGER, H., ET AL. Dynamic application-layer protocol analysis for network intrusion detection. In *Proc. of USENIX Security* (2006).

[18] DUMITRIU, D., KNIGHTLY, E., KUZMANOVIC, A., STOICA, I., AND ZWAENEPOEL, W. Denial-of-service resilience in peer-to-peer file sharing systems. In *Proc. of ACM SIGMETRICS* (2005).

[19] GEELS, D., ET AL. Friday: Global comprehension for distributed replay. In *NSDI* (2007).

[20] GUMMADI, K., ET AL. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of ACM SOSP* (2003).

[21] KILLIAN, C., ET AL. Finding liveness bugs in system code. In *NSDI* (2007).

[22] LABOVITZ, C., ET AL. Experimental study of Internet stability and backbone failures. In *FTCS: International Symposium on Fault-Tolerant Computing* (1999).

[23] LIANG, J., ET AL. Pollution in P2P file sharing systems. In *Proc. of IEEE INFOCOM* (2005).

[24] LIANG, J., ET AL. The index poisoning attack in P2P file-sharing systems. In *Proc. of IEEE INFOCOM* (2006).

[25] LIU, X., ET AL. Wids checker: Combating bugs in distributed systems. In *NSDI* (2007).

[26] LIU, X., ET AL. D3s: Debugging deployed distributed systems. In *NSDI* (2008).

[27] MAHAJAN, R., WETHERALL, D., AND ANDERSON, T. Understanding BGP misconfiguration. In *Proc. of ACM SIGCOMM* (2002).

[28] PARKER, A. The true picture of p2p filesharing. Tech. rep., july 2004. `http://www.cachelogic.com/home/pages/research/p2p2004.php`.

[29] PAXSON, V. Bro: A system for detecting network intruders in real-time. *Computer Networks 31* (1999).

[30] PROVOS, N. A virtual honeypot framework. In *Proc. of USENIX Security* (2004).

[31] SAROIU, S., GUMMADI, P., AND GRIBBLE, S. A measurement study of peer-to-peer file sharing systems. In *Proc. of Multimedia Computing and Networking* (2002).

[32] SINGH, A., ET AL. Using queries for distributed monitoring and forensics. In *EuroSys* (2006).

[33] STEINER, M., ET AL. A global view of KAD. In

Proc. of ACM/USENIX IMC (2007).
[34] YEGNESWARAN, V., ET AL. Using honeynets for internet situational awareness. In *In Proc. of ACM Hotnets IV* (2005).