Measurement and Diagnosis of Address Misconfigured P2P Traffic

Zhichun Li, Anup Goyal†, Yan Chen, Aleksandar Kuzmanovic
Northwestern University, Evanston, IL, USA †Yahoo! Inc., Sunnyvale, CA, USA

***Abstract*—Misconfigured P2P traffic caused by bugs in volunteer-developed P2P software or by attackers is prevalent. It influences both end users and ISPs. In this paper, we discover and study address-misconfigured P2P traffic, a major class of such misconfiguration. P2P address misconfiguration is a phenomenon in which a large number of peers send P2P file downloading requests to a "random" target on the Internet. On measuring three Honeynet datasets spanning four years and across five different /8 networks, we find address-misconfigured P2P traffic on average contributes 38.9% of Internet background radiation, increasing by more than 100% every year.**

In this paper, we design the $P2PScope$, a measurement tool, to detect and diagnose such unwanted traffic. After analyzing about two TB data and tracking millions of peers, We find, in all the P2P systems, address misconfiguration is caused by resource mapping contamination, *i.e.*, the sources returned for a given file ID through P2P indexing are not valid. Different P2P systems have different reasons for such contamination. For eMule, we find that the root cause is mainly a network byte ordering problem in the eMule Source Exchange protocol. For BitTorrent misconfiguration, one reason is that anti-P2P companies actively inject bogus peers into the P2P system. Another reason is that the KTorrent implementation has a byte order problem. We also design approaches to detect anti-P2P peers without false positives.

I. Introduction

Peer-to-peer (P2P) traffic has grown to be one of the dominant sources of Internet traffic Given the unprecedented amount of P2P traffic flowing through the Internet, misconfiguration, 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 [1], without compromising a single host.

The first contribution of this paper is to discover the existence and prevalence of address-misconfigured P2P traffic. Usually it is believed that the so-called "Internet background radiation" [2]—the unexpected traffic sent to all the IP addresses including unused ones—is mainly scanning traffic or DoS backscatter traffic. However, after we analyze about two TB data from three Honeynets/Honeyfarms spanning four years on five different /8 networks, we discover 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 2. Neither academia nor industry, however, has paid attention to this problem. To the best of our knowledge, we are the first to study the address-misconfigured P2P traffic.

We believe that the misconfigured P2P traffic is more prevalent than what we observe, *i.e.*, those caused by address misconfiguration. It can also be caused by port misconfiguration, firewall or NAT incompatibility. Such problems are rooted from the large number of volunteer-developed P2P software variants, which are 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 as from anti-P2P companies. Unlike content poisoning, such unwanted traffic has received little attention.

In this paper, we call for more research in studying such unwanted traffic, and diagnose its root causes. It will benefit both end users and ISPs. For end users, such unwanted traffic affects their P2P system performance and can involve innocent users into DDoS attacks unconsciously [1]. 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.

In addition, from the ISPs' perspective, such misconfigured traffic wastes resources of Internet. As the first step towards studying all unwanted misconfigured P2P traffic, 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 III-G. Therefore, given a 10Gb/s intercontinental link (*e.g.*, between LA and Tokyo) costs *$1.4 million* per year to lease [4], 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 for $detection$ and $diagnosis$ of such unwanted traffic. Here detection refers to attribute the misconfiguration to a particular variant or version of the P2P software that contains bugs, or a particular peer group, *e.g.*, peers from anti-P2P companies. This is of crucial importance, because, without doing so, nobody will take the responsibility to fix the problem. For instance, BitTorrent has more than 50 different client variants. Without knowing which one has bugs, it is hard to ask the developers of all the software

to spend time on a problem that is possibly not related to their software. Diagnosis means inferences of its root cause, *e.g.*, to locate the software code which contains the bug, or to identify the anti-P2P peers which trigger the misconfiguration.

As the first step, in this paper, we focus on detecting and diagnosing the address-misconfigured P2P traffic. We design two general principles: (1) P2P software testing by tracking its information flow in a controlled environment, and (2) P2P traffic measurement including both passive monitoring and active backtracking to identify misconfiguration events in the wild.

Applying the principles above, we design and implement *P2PScope*. Our analysis shows that *all* address misconfigurations are caused by resource mapping contamination, *i.e.*, the peers returned through indexing are invalid, thus causing data plane traffic radiation. Different systems have different reasons for contamination. We have performed root cause analysis for two largest P2P systems [5]: eMule and BitTorrent.

For eMule, one major root cause is a network byte ordering problem in the source exchange protocol. We confirm aMule (an eMule variant) has the byte order bug. For BitTorrent, address misconfiguration is mostly disseminated by the Peer Exchange (PEX) protocol implemented by uTorrent-compatible clients. We find two reasons. (i) Some anti-P2P companies actively inject bogus peers through the PEX protocol using modified Azureus. (ii) KTorrent has a byte ordering problem. We have confirmed the bug with the developers.

Furthermore, we design two approaches for anti-P2P peer detection. We validate them with the anti-P2P IP list and show they are accurate and can detect more anti-P2P IP blocks that are not in the existing anti-P2P IP list.

II. Definition

A. What is Address Misconfiguration?

P2P address misconfiguration is the phenomenon in which a large number of peers send P2P downloading requests to a "random" target on the Internet. We call such traffic address-misconfigured P2P traffic. We detect such traffic via a set of Honeynet systems that monitor unused IP space with "fake" responses. 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.

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. In this paper, we use "address misconfiguration" to refer to both the cases unless denoted otherwise.

B. Definition and Classification of Peers

To ease the discussion, we classify peers in a P2P network based on their characteristics 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.

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.

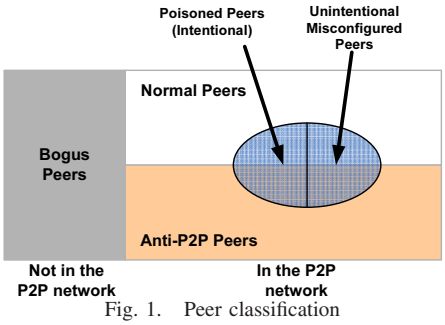

Fig. 1. Peer classification

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 [6].

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.

III. Passive Measurement Analysis

A. Data Collections

In our study, we mainly use three datasets collected at different Honeynet/Honeyfarm sensors.

	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 I
Data Description

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 [7] configured. We have data from 2004 through 2007. Honeyd is configured such that it simulates the common protocols on well-known port numbers and acts 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 the Recording Industry Association of America repeatedly complains the Honeynet IPs hosting their movies (actually false positives).

The NU Honeynet dataset The NU Honeynet sensor in Northwestern University has 10 discontinuous /24 IP blocks within three different /8 IP prefixes. It has also Honeyd configured. It uses the same Honeyd configuration as LBL Honeynet until 08/23/2007. After that, we have 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 IV-B1.

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

B. Address Misconfiguration Event Identification

To identify address-misconfigured P2P traffic, we first filter out the scanning traffic caused by botnets or worms based on whether they have malicious payloads or whether they

scan a large number of IP addresses ($\geq$ 10). We have also checked the traffic removed against P2P protocol signatures, and have not found any matches. For remaining traffic, we detect P2P connections based on whether their payloads match P2P protocol signatures. We have enhanced the P2P signatures from the L7-filter [9] and integrated them with Bro [10].

After that, we aggregate the address-misconfigured P2P traffic to events. We find all the peers target a few hotspots in Honeynets. 97% of peers only contact one Honeynet IP per day (all contact less than four 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 of the number of unique sources.

The above method 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 II for the event 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 that they are actually address-misconfigurations events, because there are no similar patterns known for Internet background radiation. The likely cases account for only 9.6% of the total events, showing that we are able to identify most cases accurately. We have manually checked the payloads of a few "likely" events. Apparently, in an event, the packets from different sources look similar, so they might come from the same P2P protocol.

C. P2P System Diversity

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

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 most 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 protocols with more than 50 different BitTorrent client software. EMule is the popular open source P2P software. Most people use eMule and its variants including aMule, xMule and eMule Mods (modifications of the original eMule). Gnutella is another famous decentralized unstructured P2P system.

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

D. Characteristics of Address Misconfiguration Events

We also discover that such events are quite heterogeneous. The scale of events (in terms of duration and the number of unique sources involved) varies substantially. Some events are as short as 1 – 3 hours; some are as long as one month. The average number of peers in eMule events is 1,404 for LBL and 1,028 for NU. BitTorrent events have smaller scale. The average number of peers in BitTorrent events is 894 for LBL and 291 for NU.

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

TABLE II
ADDRESS-MISCONFIGURED P2P TRAFFIC DISTRIBUTION AND EVENT DISTRIBUTION (LBL AND NU HONEYNET DATASETS).

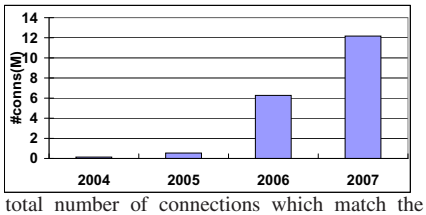

Fig. 2. The total number of connections which match the P2P payload signatures from 2004 to 2007 (LBL).

E. Prevalence in Time and Space

We use the four-year LBL data to analyze the temporal trend of address-misconfiguration. Since the scale of the events differs substantially, instead of using the number of events, we use total number of connections that 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 2 shows the increasing trend of address-misconfigured P2P traffic. We have also studied the trend on smaller scales (days and months), but due to the large fluctuation of address misconfigured P2P traffic, the trend is less obvious. Additionally, we analyze the annual trend of the total volume of the Internet background radiation in the LBL sensor. We find the yearly total numbers of connections from 2004 through 2007 are: 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 stable when compared to the growing trend of the misconfiguration traffic.

We also observe 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.

F. 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 III
PERCENTAGE ESTIMATIONS (07/2007 – 12/2007 DATA OF LBL AND NU HONEYNET DATASETS).

We have estimated the percentage of address-misconfigured P2P traffic in Internet background radiation traffic. We use the data (07/2007 – 12/2007) from the LBL and NU Honeynets for this analysis. We treat the percentage estimated in each /8 prefix of the Honeynet sensors as an independent sample. We then use the average percentage from all the 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 measure the percentages for the four /8 prefixes as shown in Table III. *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 linear extrapolation as the first order estimation, which might not be very accurate. The traffic speeds of the four prefixes pass the Chi-square Goodness of Fit test with p-value=0.9, which suggests that the traffic speeds on different prefixes might follow a uniform distribution. We use the average traffic speed of the four prefixes as a more representative speed. Since 94% of traffic seen in Honeynet is TCP traffic, and most peers run Windows, we 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 a RST packet, or if the port is open, we will see more traffic. As a conservative estimation, however, we ignored such cases. Hence, in each connection, we see 180 bytes. Based on this, we estimate that 7.90Gb/s (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.

G. Where Do Misconfigured Peers Come From?

We use the IP-to-location mapping service [11] to find the locations of misconfigured peers. For eMule, more than 67% of the misconfigured peers are from Europe. For BitTorrent, both LBL and NU data suggest that China is the place where most misconfigured peers are from (more than 90%). This confirmed the prevalence of eMule in Europe and BitTorrent in China [12].

IV. System Design

A. General Design Principles

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

Two approaches are useful for detecting the misconfiguration (attributing the problem to a particular software version or a peer group): (i) measurement including both passive monitoring and active backtracking; (ii) tracking the information flow for the suspicious P2P software in a controlled environment. The first approach helps quickly identify which software version, communication mechanism or peer groups are potentially 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.

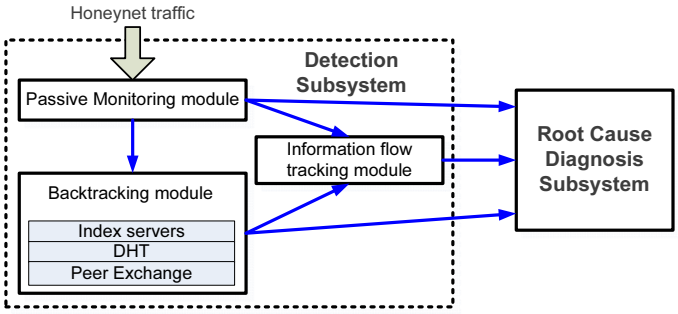

Fig. 3. Architecture of the P2PScope system.

After detection, we want to further diagnose the root causes of the misconfiguration. This diagnosis step is harder than detection, and usually requires manual inspection. Information flow tracking is still very useful in this stage. Another useful approach is to form the hypotheses regarding the root causes and then to conduct experiments for validation.

Per the principles above, P2PScope has two major subsystems: detection subsystem and diagnosis subsystem, as shown in Figure 3. P2PScope are capable of real-time monitoring and diagnosis, *i.e.*, both the detection subsystem and the diagnosis subsystem can work in real time.

B. Detection Subsystem

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, the 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 the information flow tracking module. In the information flow tracking module, we install that version of P2P software and check its peer propagation. Finally, all collected information will be output to root cause diagnosis subsystem.

1) Passive Monitoring Module: We find that Honeynets are useful for monitoring the address-misconfigured P2P traffic on unused IP blocks. The Honeynets respond to the incoming connections and generate "faked" responses. We use Honeyd-based [7] lightweight Honeynets. Simulating 10 /24 networks on a Pentium4 3GHz machine uses on average less than 5% of CPU and around 40MB of memory.

Honeyd uses port numbers to identify protocols, but P2P traffic is on random port numbers. To overcome this limitation, we apply a *payload signature based* approach. None of the existing tools supports such an option. We modify 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.

2) Backtracking Module: To understand how the peers get misconfigured, the backtracking module tracks the peers in real-time. There are several challenges for designing and implementing the backtracking module: (i) P2P systems have multiple ways to obtain peers for a file, such as index servers, DHT and peer exchange protocols; (ii) some P2P systems, such as BitTorrent, have multiple incompatible implementations for certain protocols; (iii) some P2P software and protocols are either closed-source or lack of documentation. We try our best to implement modules for each of these protocols.

We implement lightweight eMule and BitTorrent clients to query multiple index servers simultaneously. 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 also implement peer exchange protocols (query more peers from known peers) and DHT for BitTorrent and eMule.

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

3) Information Flow Tracking Module: Whenever a certain P2P client version is suspicious, we install it in our controlled monitoring environment. In the controlled monitoring 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 that never comes in, we know the software will cause address misconfiguration.

C. Diagnosis Subsystem

Tracking down the root causes 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. (ii) Trace the Honeynet IP addresses propagated in the P2P systems. (iii) Check the routability of the peers returned. (iv) Check whether the peers are on the anti-P2P IP lists and analyze their behavior. (v) Check the reverse byte order of peer IPs, because it is one of the most frequent mistakes in networked system implementations.

Most of events are caused by BitTorrent and eMule. Thus, our current diagnosis focuses on these two. 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 apply the taint check techniques to trace how the bogus peers are produced.

Another useful toolkit is the routablity checking. In [13], Cooke *et al.* discover that 66.8% of all possible IPv4 addresses are unroutable. If we assume that the bogus peers are produced randomly, we would find a reasonably large percent of unroutable bogus peers. If the percentage is small, it implies that bogus peers are not produced 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 the anti-P2P IP list. We compile the anti-P2P IP list from several resources: (i) the anti-P2P blacklist software Peer Guardian [14], and (ii) various P2P discussion forums. The list might not be fully comprehensive, but it 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 bogus peers in misconfigured P2P networks? (v) how badly are individual clients affected by misconfigurations?

V. Detection & Diagnosis Results

We have deployed P2PScope in the NU Honeynet for real-time monitoring and diagnosis since August 2007.

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 address-misconfigured P2P traffic.

A. 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 propagate quickly in P2P systems and cause large amount of address-misconfigured P2P traffic.

B. eMule Misconfiguration Diagnosis

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, among 1,771,296 misconfigured peers we observed, 99.90% of peers are normal peers. Therefore, we conclude that *normal peers are mainly involved in the address misconfiguration of eMule.*

In addition, the files involved in an eMule event are 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 any specific file shared in the eMule network.

2) What Is the Root Cause: When analyzing the network level behavior of the eMule misconfiguration, we notice that the IP addresses formed by reversing the byte order of the targets were mostly alive. 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 by locating the bug in source code.

More details of the experiments are in our technique report [15]. In the first experiment, we test 13 targeted destination

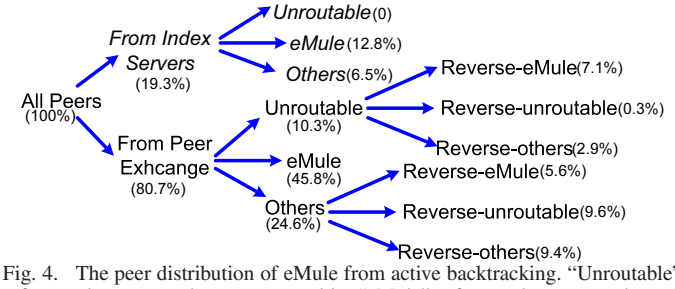

Fig. 4. The peer distribution of eMule from active backtracking. "Unroutable" refers to those peers that are unroutable; "eMule" refers to those peers that are running eMule which is verified by probing them with eMule protocol; and "others" are all the remaining routable peers. "Reverse" means we study its reverse IP.

in real-time, and have observed that 61% of reverse IPs are indeed running eMule with the same port number of the targets. In the second experiment, we check whether the reverse IP addresses of unroutable peers observed in the peer list of a file hash run eMule. We aggregate the result of 100 file hashes found in real-time. As shown in Figure 4, 10.3% of backtracked peer IPs are unroutable. Among them, we observe 69.6% whose reverse IPs run eMule. This, again, is another strong evidence in favor of the byte ordering hypothesis.

We further locate the bug in the source code of aMule, a popular eMule alternative. EMule has a complex client ID to IP mapping system called HybridID. 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; thus it causes the byte order bug. This example shows that the byte ordering problem sounds simple but it can sometimes get really complex. Due to the complexity of the HybridID system, it is highly likely that other eMule alternatives might have similar bugs.

3) How Is the Misconfiguration Disseminated: EMule offers three ways to obtain peers for a file hash: index servers, eMule source exchange protocol, and eMule Kademlia DHT. We would like to know which one disseminates the misconfiguration. Since we have not observed any misconfigured DHT traffic, we believe the DHT does not have the byte ordering problem. To attribute the problem to index servers or the source exchange protocol, we have queried 100 file hashes and got a total of 37,079 unique peers. We find that 10.7% of the peers from source exchange have Honeynet IPs in their neighbor list, but the index servers never return Honeynet IPs. We have also done a routability check. 12.8% of peers returned by source exchange protocol are unroutable; none of the peers returned by index servers is 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* an index server but the source exchange protocol which is used for disseminating the misconfiguration.

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) that have the byte ordering problem. This is challenging because there are certain cases in which we cannot determine whether they have a byte ordering problem. Instead, we estimate its lower and upper bound and make the bounds as tight as possible.

As show in Figure 4, if a peer is not running eMule but its reverse IP is, we believe it definitely has a byte ordering problem. Such peers are 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 also 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.

5) How Badly Are Individual Clients Affected by Misconfigurations: Similarly, based on the measurement of peers' neighbor lists, we estimate, the average of the bogus neighbor percentage of a peer is in the range of [7.4%, 22.8%]. More details are in our technique report [15]. Among all the peers, 6.3% of peers are heavily influenced by misconfiguration. For these clients, more than 80% of peers in their neighbor lists are unroutable.

C. BitTorrent Misconfiguration Diagnosis

1) Which Peers Spread Misconfiguration: When analyzing the historical events discussed in Section III-C, we have found two groups of BitTorrent events with exclusive distinct network-level characteristics.

For one group of events, the incoming sources are mainly from a small number of /24 networks that belong to co-location server farm providers, such as XEEX. Through further study, we find these IP blocks belong to anti-P2P companies, such as Media Defender, MediaSentry corp., Net Sentry *etc.*Therefore, we know this group of events is anti-P2P related.

Furthermore, for all the events, we calculate the ratio of anti-P2P peers to the total peers based on the anti-P2P IP list. 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. Figure 5 shows the percentage of anti-P2P peers in each event from the NU Honeynet. The x-axis is event ID. The results from LBL Honeynet are similar. The events dominated by anti-P2P peers surprise us, since they show that *the anti-P2P companies themselves somehow get misconfigured solely*.

We call anti-P2P peer dominated events *anti-P2P peer events* and normal peer dominated *normal peer events*. We have 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.

As shown in Table IV, we compare the properties of these two types of events and discover that their characteristics are indeed different. Peers from anti-P2P peer events all use Azureus. We conjecture that anti-P2P companies modify Azureus 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 run on a diverse set of clients. In addition, the peers in anti-P2P peer events are highly coordinated. They start contacting the Honeynet IPs in almost the same time. They also depart in the same

	Normal peer events	*Anti-P2P peer events*
# of events	136-NU, 36-LBL	75-NU, 39-LBL
Client Software- NU	90% - UTorrent 10% -Others	100% - Azureus
Client Software- LBL	57% - Bit Spirit, 31% - BitComet 12% -Others	100% - Azureus
Message Length	Variable lengths	Similar lengths
Files Queried	Diverse	Latest music/movies
Avg. 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 # of /24

TABLE IV
COMPARISON BETWEEN NORMAL PEERS EVENTS AND ANTI-P2P PEER EVENTS.

time. By contrast, the peers in normal peer events arrive and depart gradually. The messages sent in anti-P2P peer events are almost same. They have a similar message length and even a similar bitmap that annotates the content availability distribution. To some extent, the group of anti-P2P peers is 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. However, the normal peer events are little coordinated.

2) How to Detect Anti-P2P Peers: Besides using the anti-P2P IP list collected online, we design two behavioral anti-P2P peer detection approaches: one is based on Honeynets/Honeyfarms, and the other can be applied by any single client.

Approach I has the following two steps: (i) detect anti-P2P peer events; and (ii) in the anti-P2P events, detect the anti-P2P IP subnets that contain the anti-P2P peers. To separate anti-P2P peer events from normal peer events, a good indicator is the ratio of the number of unique /24 subnets vs. the number of unique IP addresses. We call it the U metric. We use $U = 0.5$ as the threshold. When $U > 0.5$, we consider the event as a normal peer event, and otherwise an anti-P2P peer event. Figure 6 shows the U metric distribution in NU BitTorrent events. It indicates that the U metric is not sensitive to the threshold. We also test $U = 0.5$ on the LBL dataset and obtain the exactly same result as we use the anti-P2P IP list. Then, in a detected anti-P2P event, we treat any /24 subnets with more than five unique IPs observed as the IP blocks from anti-P2P companies. We evaluate the 18 subnets we detected. They are either in the anti-P2P IP list or they belong to server farm providers such as XEEX. We believe they are all anti-P2P subnets.

Approach II can be used to detect anti-P2P peers by any single client. Anti-P2P peers want to maximize their damage. We conjecture that they might reply to any file hash including non-existing ones. We find this indeed the case as shown in Section V-C3. We leverage such behavior to detect anti-P2P peers. When a client wants to check a peer, the client can just send a peer exchange message with a fake file hash generated randomly. The chance that a randomly generated 160bit file hash exists is almost zero. If the peer claims that it has the fake file hash, we know it is an anti-P2P peer. This detection will not have any false positives, since normal peers will not

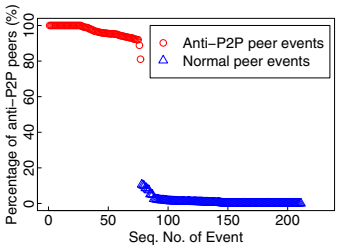

Fig. 5. Percentage distribution of anti-P2P peers in NU BitTorrent events.

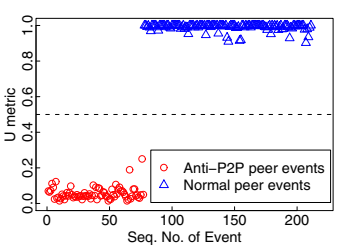

Fig. 6. The distribution of U metric in NU BitTorrent events

respond to any non-existing file hash. By using this approach, just from the neighbor lists of misconfigured peers, we have successfully detected more than 100 anti-P2P peers that are not in the anti-P2P IP list we have.

3) What Is the Root Cause: We have not captured any real-time anti-P2P peer 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 bogus peers in the P2P systems. Our P2PScope system tracks the misconfigured peers in real-time. Using the approach II developed in the previous section, we detect more than 100 anti-P2P peers that are not in the anti-P2P IP list. These anti-P2P peers have more than 50% unroutable IPs in their neighbor lists. All of these anti-P2P 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 discover that each of these IPs runs 100's of BitTorrent clients simultaneously. The IPs also support VMWare authentication protocol. We conjecture the anti-P2P company uses VMware based virtual machines to run multiple BitTorrent clients. All of the clients run a version of Azureus 2500. Moreover, as detected by approach II, these anti-P2P clients respond to any arbitrary file hash. Artistdirect might have modified the Azureus client for their purpose.

We have checked the neighbor lists returned by these anti-P2P IPs. Most returned peers are 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. We suspect anti-P2P companies want to slow down the downloading process by supplying bogus peers randomly.

The *anti-P2P related normal peers* are the normal peers that query the file hashes anti-P2P peers are interested in. Most of these peers ($> 90\%$) are within two hops to anti-P2P peers. 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*. Since it is impossible to identify all the anti-P2P peers, this estimation is quite conservative.

Root Cause II: the byte order problem of KTorrent clients. We further backtrack the file hashes requested by the misconfigured peers. We find a large portion of unroutable peers that we get are supplied by the KTorrent peers. Therefore, we believe KTorrent is suspicious. We also study UTorrent and Azureus, since they are the most popular clients. We set up a controlled BitTorrent farm on three machines, and each runs one of the three clients with default configuration. We test top seven

```
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);
}
```

Fig. 7. Code Snippet of KTorrent Byte Order Bug

torrent files seen in Honeynets.

We classify the peers seen by our clients into two types: (i) *incoming peers*—peers that come to the clients through trackers (index servers in BitTorrent terminology), Peer Exchange (PEX) messages, DHT, and the sources of incoming connections; (ii) *outgoing peers*—peers that are sent out by the clients through PEX messages and DHT. *outgoing - incoming* peers are the peers that go out from the client but never come in. Those peers are the bogus peers created by the client itself. In Table V, we have shown the statistics of three clients. Among all outgoing KTorrent peers, 56% of them are bogus. Thus, it shows that KTorrent is one of the sources of misconfiguration.

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

TABLE V
PEX IP PROPAGATION STATISTICS OF DIFFERENT BITTORRENT CLIENT.

After that, we study the source code of KTorrent and pinpoint the exact bug. The problem is also a byte ordering problem. In Figure 7, we show the code snippet related to this bug. The UTorrent PEX protocol requires the IP addresses in peer exchange messages to be in network byte order. When KTorrent sends out the peer exchange messages, 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 that returns IP addresses in network byte order. However, in *WriteUint32* function implemented by KTorrent, it reverses the byte order again, and thus causes the problem. To our knowledge, we have not found anyone else reports this bug.

4) How Is the Misconfiguration Disseminated: Similar to the eMule study, we find that BitTorrent DHT is unlikely to spread or to generate misconfigurations.

Unlike eMule, BitTorrent has several incompatible peer exchange protocols proposed by popular BitTorrent clients, such as 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 protocols. The question is whether the misconfigurations are propagated through trackers or PEX protocols? And if the latter, through which PEX protocol?

Similar to the case study of eMule, we study which protocols can return honeynet IPs. The result shows that only uTorrent PEX protocol returns the honeynet IPs. Moreover, all misconfigured peers observed from Honeynets are uTorrent PEX compatible clients, and most of them are actually uTorrent. See [15] for more details.

To understand which uTorrent PEX compatible clients propagate the misconfigurations, we study the behavior of different clients. We mainly study, when a client obtains peers from uTorrent PEX protocol, how it checks the availability of the peers before it further propagates the peers. We classify the behavior into three categories: (i) no checking, (ii) requiring peers to respond SYN-ACK (only active port and not running BitTorrent), and (iii) requiring peers respond to BT-HANDSHAKE messages. KTorrent propagates anything that comes to it without even checking for an open port. UTorrent propagates anything that has an open port. Azureus only propagates peers that are active BitTorrent clients. 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.

5) What Is the Percentage of Bogus Peers in the Misconfigured P2P Networks: We check 9,000 backtracked peers, we find 0.14% are unroutable and 35% of them cannot be connected. Therefore the percentage of bogus peers is within [0.14%, 35%].

6) How Badly Are Individual Clients Affected by Misconfigurations: Similar to eMule study, we observe 9,834 misconfigured peers from Honeynets. We measure how many bogus peers are in their neighbor lists. We find the average percentage is within [0.053%, 32%].

D. Validation with External Misconfiguration Events

Through the validation with the misconfiguration events observed without using Honeynets, we believe our findings are representative. We study 15 popular files; the result suggests that the bounds of bogus peer percentage are similar as we report before. See [15] for more details.

We also study whether the misconfigured P2P traffic (in particular, by normal peers) is different than normal P2P traffic in terms of the network level characteristics. We find they are almost identical. Hence, filtering based on network-level characteristics is not applicable. See [15] for more details.

VI. 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 [16]. There has also been much recent work that considers the various instabilities and misconfiguration in BGP [17]. A study of firewall misconfiguration [18] 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 misconfiguration.

P2P measurement: Most existing measurement studies of P2P networks are based on the measurement of the *normal* P2P traffic [19], [20]. On the other hand, we measure the

abnormal P2P traffic observed in Honeynets. This is also observed by Yegneswaran *et al.* in [2]. 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-misconfigured P2P traffic, and its root causes. In [21], it is found that pollution (both intentional and unintentional) is wide spread for popular files. In [22], two other pollution designs are proposed. In [23], 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 [6]. 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 [24], [25], online monitoring [26], [27], log forensic analysis [28], [29], network trace based inference [30], and model checking [31]. Although some of the approaches might be useful for the root cause diagnosis of P2P address misconfiguration, the following properties make leveraging any of the above existing techniques hard. First, the P2P systems that we diagnose are heterogeneous. 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 difficult to log the network traces from remote peers. Third, the P2P systems that 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 a light weight approach to diagnose the address misconfiguration.

VII. Conclusions

In this paper, we report the prevalence of address misconfiguration. The address-misconfigured P2P traffic is 38.9% of the Internet background radiation and is growing quickly. Since such traffic is harmful for both end users and ISPs, we believe the problem needs to be understood and solved.

To detect such misconfiguration and to understand the root causes, we have designed the P2PScope system. Using this system, we find that the data plane traffic radiation is the common characteristic of the six P2P systems. For eMule, the problem is mainly caused by a byte ordering problem. For BitTorrent, we find two reasons: (i) anti-P2P companies deliberately injecting invalid peers; (ii) KTorrent byte ordering bug. Given new software bugs or malicious motives likely appear in future, we need P2PScope to monitor address misconfiguration continuously.

VIII. Acknowledgments

Our thanks to Vern Paxson and the Lawrence Berkeley National Laboratory for support of the LBL honeynet operations. This work was supported in part by NSF Awards 0627715, DoD Young Investigator Award FA9550-07-1-0074, and DoE Career award DE-FG02-05ER25692/A001. Opinions, findings, and conclusions or recommendations are those of the authors and do not necessarily reflect the views of the funding sources.

References

[1] "Peer-to-peer networks co-opted for DOS attacks," http://www.securityfocus.com/news/11466.
[2] V. Yegneswaran et al., "Using honeynets for internet situational awareness," in *In Proc. of ACM Hotnets IV*, 2005.
[3] "List of bittorrent clients," http://en.wikipedia.org/wiki/List_of_BitTorrent_clients.
[4] "Global bandwidth research service executive summary," http://www.telegeography.com/products/gb/pdf/Executive_Summary.pdf.
[5] Andrew Parker, "The true picture of p2p filesharing," Tech. Rep., july 2004, http://www.cachelogic.com/home/pages/research/p2p2004.php.
[6] J. Liang et al., "The index poisoning attack in P2P file-sharing systems," in *Proc. of IEEE INFOCOM*, 2006.
[7] N. Provos, "A virtual honeypot framework," in *Proc. of USENIX Security*, 2004.
[8] W. Cui et al., "GQ: Realizing a system to catch worms in a quarter million places," Tech. Rep. TR-06-004, ICSI, 2006.
[9] "Application Layer Packet Classifier for Linux," http://l7-filter.sourceforge.net.
[10] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, 1999.
[11] "Ip to asn mapping," http://www.team-cymru.org/Services/ip-to-asn.html.
[12] M. Steiner et al., "A global view of KAD," in *Proc. of ACM/USENIX IMC*, 2007.
[13] E. Cooke et al., "The dark oracle: Perspective-aware unused and unreachable address discovery," in *Proc. of USENIX NSDI*, 2006.
[14] "Peer Guardian, an IP blocker for p2p system," http://phoenixlabs.org/pg2/.
[15] Z. Li, A. Goyal, Y. Chen, and A. Kuzmanovic, "P2p doctor: Measurement and diagnosis of misconfigured peer-to-peer traffic," Tech. Rep. NWU-EECS-07-06, Northwestern University, 2009.
[16] Craig Labovitz et al., "Experimental study of Internet stability and backbone failures," in *FTCS: International Symposium on Fault-Tolerant Computing*, 1999.
[17] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proc. of ACM SIGCOMM*, 2002.
[18] F. Cuppens et al., "Detection and removal of firewall misconfiguration," in *International Conference on Communication, Network and Information Security*, 2005.
[19] K. Gummadi et al., "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. of ACM SOSP*, 2003.
[20] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. of Multimedia Computing and Networking*, 2002.
[21] J. Liang et al., "Pollution in P2P file sharing systems," in *Proc. of IEEE INFOCOM*, 2005.
[22] N. Christin et al., "Content availability, pollution and poisoning in file sharing peer-to-peer networks," in *Proc. of ACM Electronic Commerce*, 2005.
[23] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel, "Denial-of-service resilience in peer-to-peer file sharing systems," in *Proc. of ACM SIGMETRICS*, 2005.
[24] X. Liu et al., "Wids checker: Combating bugs in distributed systems," in *NSDI*, 2007.
[25] D. Geels et al., "Friday: Global comprehension for distributed replay," in *NSDI*, 2007.
[26] A. Singh et al., "Using queries for distributed monitoring and forensics," in *EuroSys*, 2006.
[27] X. Liu et al., "D3s: Debugging deployed distributed systems," in *NSDI*, 2008.
[28] M. K. Aguilera et al., "Performance debugging for distributed systems of black boxes," in *ACM SOSP*, 2003.
[29] P. Barham et al., "Using magpie for request extraction and workload modelling," in *OSDI*, 2004.
[30] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *ACM SIGCOMM*, 2007.
[31] C. Killian et al., "Finding liveness bugs in system code," in *NSDI*, 2007.