A Measurement Study of Attacks on BitTorrent Seeds

Prithula Dhungel[†], Xiaojun Hei[‡], Di Wu[§], Keith W. Ross[†]
[†]Polytechnic Institute of NYU, Brooklyn, NY 11201
[‡]Huazhong University of Science and Technology, P. R. China
[§]Sun Yat-Sen University, P. R. China
Email: pdhung01@students.poly.edu, heixj@hust.edu.cn, wudi27@mail.sysu.edu.cn, ross@poly.edu

***Abstract*—We study a natural and potentially devastating attack against BitTorrent, namely, attacking the initial seed in a torrent's early stages. The goal of this attack is to diminish the seed's ability to upload blocks. If the attacker can discover and react quickly enough to the new torrent, it can possibly "nip the torrent in its bud," preventing all of the leechers from obtaining the entire file. We consider two natural seed attacks: the *bandwidth attack* and the *connection attack*. We take a three-prong approach to analyze these attacks. First, we actually launch and measure the attacks using popular BitTorrent seeds (Azureus, uTorrent, and BitTornado). To this end, because we do not want to interfere with torrents in the wild, we have created our own private torrents within PlanetLab. Second, to gain insight into our empirical results, we carefully analyze the connection management and seeding algorithms in open-source BitTorrent seeds. Third, we construct a simple fluid model which provides additional insights into the empirical results. We have discovered that the three BitTorrent seeds investigated are quite resilient to such an attack. The observations and conclusions in this paper can help P2P developers design highly-resilient P2P systems.**

I. Introduction

Over the past several years, the music industry has been aggressively attempting to curtail the distribution of targeted musical recordings over P2P file sharing networks. Today, BitTorrent is one of the most popular P2P file distribution protocols, particularly for the distribution of large files such as high-definition movies, television series, record albums, and open-source software distributions. Not surprisingly, the music, film, and television industries have begun to hire anti-P2P companies to curtail the distribution of "assets" in BitTorrent. These interdiction attacks represent some of the systematic attacks taking place in the Internet today.

It is important to study how vulnerable the BitTorrent ecosystem is to large-scale, resource-intensive attacks for the following reasons. (i) It is critical that we design future generations of P2P file distribution protocols that are highly resilient to attacks. (ii) Understanding BitTorrent's vulnerabilities will bring insight into the vulnerabilities of broader classes of P2P systems like live streaming and video on demand, for which BitTorrent serves as a blueprint.

In this paper we present a preliminary study on a natural and potentially devastating attack, namely, attacking the initial seed in the early stages of a torrent. (A comprehensive report can be found at [1].) The goal here is to diminish the seed's ability to upload blocks. If the attacker can discover and react quickly enough to the new torrent, it can possibly "nip the torrent in its bud," preventing all of the torrent's peers from obtaining the entire file. We consider two natural seed attacks: the *bandwidth attack* and the *connection attack*. Our contributions are as follows. (i) We launch and measure seed attacks against popular BitTorrent seeds (Azureus, uTorrent, and BitTornado). In designing the attack experiments, we didn't want to interfere with existing torrents in the wild, but nevertheless wanted to attack real BitTorrent seed implementations. To this end, we created our own private torrents on PlanetLab nodes and used this environment to attack Azureus, uTorrent, and BitTornado seeds. (ii) We have instrumented a BitTorrent attack client by modifying the code of BitTornado, and have developed a BitTorrent traffic analyzer, enabling us to track the rate at which the seed transmits to the leechers and attackers during the attack. (iii) To gain further insight into the seed attacks, and to guide our attack scenario planning, we construct a simple fluid model for attacks against seeds. The model takes into account seed upload bandwidth, the leechers and attackers download bandwidths, the number of leechers and attackers, and the seed's limit on upload slots.

We end this section with a summary of the related work. In Section II we describe the two types of seed attacks: bandwidth attack and connection attack. We describe the experimental setup for our attack experiments in section III. In Sections IV and V we present and analyze the results obtained for various experiments involving bandwidth and connection attacks. Section VI concludes the paper.

A. Related Work

Although significant progress has been made in understanding the strengths and limitations of BitTorrent's protocol and tit-for-tat mechanism [2], to date, there has been little work on its vulnerabilities to attacks. El Defrawy et. al [3] discusses how BitTorrent can be used to launch DDoS attacks against any host in the Internet. Nikitas et. al [4] analyzed the robustness of BitTorrent against selfish peers who try to download more than their fair share of data and found that BitTorrent is quite robust against such attacks. A simulation analysis of BitTorrent examined two attacks: an attack involving tampered buffer maps, and a connection monopolization attack [5]. In a previous paper [6] we investigated ongoing attacks on BitTorrent leechers and found them to be highly resilient. To the best of our knowledge, this paper is the first study of attacks on BitTorrent seeds.

II. Attacks on Initial Seeds: Taxonomy

Each leecher and seed uses one of the many BitTorrent clients. More than 50 BitTorrent clients have been developed to date. We have observed that many clients deviate significantly from Bram Cohen's Mainline client. Therefore, in drawing conclusions about BitTorrent behaviors and vulnerabilities, it is important to use popular real-world clients, rather than simulations of idealized clients.

When attacking initial seeds, the attacker attempts to diminish the seeds' ability to upload blocks (and ideally stop the seeds from uploading altogether). If the attacker can discover and react quickly enough to the new torrent, it can possibly "*nip the torrent in its bud*," preventing all of the torrent's peers from obtaining the entire file. The attacker largely has two options available: bandwidth attacks and connections attacks.

In the **bandwidth attack**, attackers exploits the vulnerability of BitTorrent seeding algorithm and attempts to clog the seed's upload bandwidth, which is typically much less than its download bandwidth due to its asymmetric access (for DSL, cable, and fiber-to-the-home). In the traditional *bandwidth-first* seeding algorithm, the seed uploads (unchokes) to at most n leechers where n is typically set to 4 or 5. The seed measures the rate at which it uploads to these n unchoked leechers. Periodically (typically every 30 seconds), it chokes the leecher to which it is uploading at the slowest rate, and replaces it with a randomly selected leecher (optimistically unchokes). In the **bandwidth attack**, the attackers detect the torrent in its early stages and connect to the seed and try to continuously occupy most of the seed's unchoke slots by downloading at a faster rate than legitimate peers.

A seed has about 50 connection slots (typically configurable). Denote c as the slot number. The seed maintains TCP connections with up to c peers, with up to n of the c peers being unchoked for upload. We refer to attacks that attempt to consume the majority of the seed's connection slots as **connection attacks**. To be successful, the attacker needs to detect the torrent in its early stages and grab the majority of the connection slots, leaving few connection slots for legitimate peers. To ensure that seeds maintain the connections, the attackers may need to occasionally download blocks.

III. Methodology

Our attack environment consists of a single seed sharing a file, 30 (legitimate) leechers, a single tracker, and a number of attack peers. The seed is located on our university network. The tracker, 30 leechers, and attackers are located on different PlanetLab nodes.

We created the attacker by modifying the source code for BitTornado 0.3.17. The attacker works by making TCP connection to the seed, downloading blocks from it, and never forwarding blocks to any other peers. The leechers and the tracker used the BitTornado 0.3.17 client. The leechers were capped with a maximum upload bandwidth of 48 KB/sec and a maximum download bandwidth of 192 KB/sec (unless otherwise stated). When selecting nodes in PlanetLab, we were careful in choosing nodes that had bandwidths enough to meet the bandwidth settings as specified. For experiments involving the bandwidth attack, no bandwidth limitations were set for attackers. Since the main idea behind the connection attack is to hinder file distribution without using a lot of bandwidth, for experiments involving connection attack, attackers were capped with a maximum upload bandwidth of 10 KB/sec and a maximum download bandwidth of 20 KB/sec. Table I summarizes the parameter settings of the experiments.[1] As part of the attack environment, we developed a BitTorrent traffic analyser that enables to determine the amount of bandwidth the seed provides to the leechers and attackers over time.

We attacked three different seeds - Azureus (Vuze) 3.1.1, uTorrent 1.7.7, and BitTornado 0.3.17. Azureus and uTorrent are among the most popular seeds in the wild today, whereas Azureus and BitTornado are open source, which greatly facilitates analysis. Because uTorrent had the results similar to BitTornado, in this paper we only present results for Azureus and BitTornado. All the experiments simulated a flash crowd effect, in that, 5 leechers are started first; after all 5 leechers are connected to the seed, all the attackers are started, followed by the remaining 25 leechers. In these experiments, we are exploring an advantageous scenario for the attackers, in which they quickly detect and join the new torrent. For each experiment, the number of unchoke slots and maximum number of connections at the seed were set to 4 and 50 respectively.

IV. Bandwidth Attack against Initial Seeds

A. Fluid Model for Bandwidth Attack

At any given instant of the time, the seed will have TCP connections to both legitimate leechers and attackers. Let N_A denote the number of connections to attack peers, and N_L denote the number of connections to legitimate peers. Suppose all the attackers have the same downstream bandwidth of b_A; and all the legitimate peers have the same downstream bandwidth of b_L. For a bandwidth attack, it is reasonable to assume that $b_A > b_L$. Denote the seed upstream bandwidth by u and the seed's maximum number of upload slots by n. At any given time, $n-1$ slots are used by regular unchoked peers, and 1 slot is used by the current optimistic unchoked peer. First observe that each unchoked legitimate peer receives bits from the seed at a rate bounded by $\min\{b_L, u/n\}$; similarly, each unchoked attack peer receives bits from the seed at a rate bounded by $\min\{b_A, u/n\}$. There are two cases:

- **Scenario I** ($b_A > b_L \geq u/n$): In this case, each unchoked peer receives at rate u/n, so the attackers have no advantage over legitimate peers in the Bandwidth First seeding algorithm. Assuming that peers are chosen at random from the $N_L + N_A$ peers for optimistic unchoke, the long-run fraction of seed bandwidth obtained by the attackers is determined by the fraction of attackers among connected peers, that is, $N_A/(N_L + N_A)$, and the amount bandwidth available to legitimate peers is $uN_L/(N_L + N_A)$.

[1] BT - BitTornado, AZ - Azureus, b/w - bandwidth, Conn. - connection; I/II - Scenario I/II, u/l - unlimited

Type	File Size (MB)	Attacker				Leecher				Seed			
		Client	# of Attackers	b/w (KB/sec)		Client	# of Leechers	b/w (KB/sec)		Client	Unchoke Slots	Max Conn.	Up b/w (KB/sec)
				Up	Down			Up	Down				
b/w (I)	100	BT	40	u/l	u/l	BT	30	48	192	AZ	4	50	60/100/140
b/w (II)	100	BT	40	u/l	u/l	BT	30	20	40	AZ	4	50	500
Conn.	500	BT	800	10	20	BT	30	48	192	AZ	4	50	60/100/140
b/w (I)	500	BT	40	u/l	u/l	BT	30	48	192	BT	4	50	50/100/150
b/w (II)	100	BT	40	u/l	u/l	BT	30	20	40	BT	4	50	500
Conn.	500	BT	800	10	20	BT	30	48	192	BT	4	50	50/100/150
Combined	100	BT	800	u/l	u/l	BT	30	20	40	BT	4	50	500

TABLE I: Parameter Settings of the Seed Attack Experiments

- **Scenario II** ($b_A \geq u/n > b_L$): Here an unchocked legitimate peer receives at rate b_L where each unchocked attack peer receives at least at rate u/n. Thus, under the Bandwidth First seeding algorithm, all the regular unchoked slots are taken by attackers. Legitimate peers can only get optimistically unchoked. The fraction of seed upstream bandwidth obtained by attackers is thus at least $\frac{n-1}{n}$, and the amount of bandwidth available to legitimate peers is at most b_L.

This model is useful in guiding the design of our attack experiments and interpreting the measurement results.

B. Attacking Azureus

We now investigate the effectiveness of bandwidth attacks against the Azureus seed. Table II shows the results. In this table, "*Delay Ratio*" is the ratio of the average time taken to download the file at the 30 leechers in the presence of attackers to that without attackers, with other experiment settings the same. We observe that even with the number of attackers as high as 40 and the seed bandwidth as low as 60 KB/sec, the delay ratio is less than 3.

Seed Uplink (KB/sec)	Avg Download Time w/o Attackers (mins)	Delay Ratio
60	36	2.24
60	36	2.58
100	35	1.37
100	35	1.45
140	34	1.08
140	34	1.04

TABLE II: Bandwidth Attack Results for Azureus (40 Attackers; 100 MB File; Leechers: 48 KB/sec & 192 KB/sec; Attackers: No Bandwidth Cap)

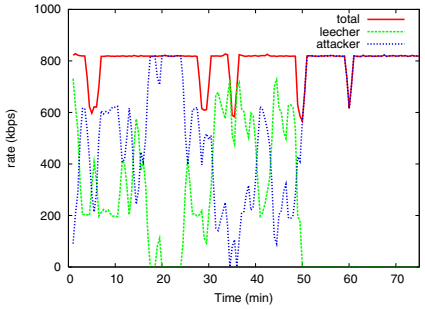

Fig. 1: Seed Bandwidth Distribution for Attackers and Leechers in an Azureus Bandwidth Attack

Fig. 1 shows the bandwidth share between attackers and leechers at Azureus seed (100 KB/sec) in a typical bandwidth attack experiment (Table II). We observe that neither the attackers nor the leechers monopolize the upload bandwidth of the seed. After the leechers finish downloading at around 50 minutes, all the bandwidth is occupied by the attackers.

To interpret the empirical results, we examined the source code of Azureus (Vuze) 3.1.1.0 to understand its seeding algorithm. The peers in the current unchoke list that have higher download rates as well as fewer bytes downloaded from the seed will be near the top of the new unchoke list. The peer at index n in this new list is then choked to make room for the optimistic unchoke.

Even if the attackers download from the seed at a high rate, because they will have downloaded more bytes already, the leechers still get unchoked at regular intervals, resulting in a low delay ratio. Furthermore, all the bandwidth experiments in Table II fall in the first scenario in the fluid model ($b_A > b_L \geq u/n$), for which attackers do not have any advantage over the leechers when fighting for unchoked slots.

Table III shows the delay results for the cases that fall into the second scenario ($b_A \geq u/n > b_L$). It can be observed that even with 40 high bandwidth attackers, the delay ratio is less than 1.5. Even though the attackers have much higher bandwidths compared to the leechers, the second unchoke criterion in the seeding algorithm – lesser amount of data downloaded so far – ensures that the leechers are able to grab unchoke slots from time to time.

Seed Uplink (KB/sec)	Avg Download Time w/o Attackers (mins)	Delay Ratio
500	77	1.29
500	77	1.13

TABLE III: Bandwidth Attack Results for Azureus (40 Attackers; 100 MB File; Leechers: 20 KB/sec & 40 KB/sec; Attackers: No Bandwidth Cap)

C. Attacking BitTornado

Seed Uplink (KB/sec)	Avg Download Time w/o Attackers (mins)	Delay Ratio
50	177	4.11
50	177	4.44
100	176	1.95
100	176	1.39
150	167	1.08
150	167	1.06

TABLE IV: Bandwidth Attack Results for BitTornado (40 Attackers, 500 MB File; Leechers: 48 KB/sec & 192 KB/sec; Attackers: No Bandwidth Cap)

Table IV shows the results for bandwidth attacks against the BitTornado seed. Once again, the attacks are not effective. For all experiments, before the leechers complete their downloads, the seed bandwidth is shared more or less equally among the attackers and leechers.

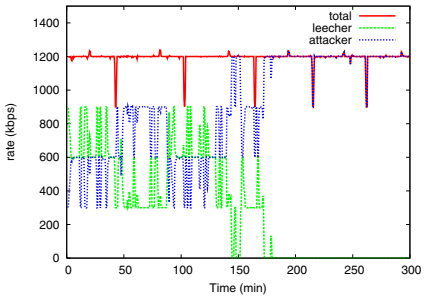

Fig. 2: Seed Bandwidth Distribution for Attackers and Leechers in a BitTornado Bandwidth Attack

As further shown in Fig. 2, for the seed bandwidth of 150 KB/sec (Table IV), despite their high download capacity, the attackers cannot take advantage over the leechers in achieving high download rates.

BitTornado uses a pure bandwidth first algorithm when seeding. However, the bandwidth settings in all experiments in Table IV fall into the first scenario in the fluid model ($b_A > b_L \geq u/n$). Hence, the attackers have no advantage over the leechers when fighting for unchoke slots at the seed and hence the low delay ratios.

Seed Uplink (KB/sec)	Avg Download Time w/o Attackers (mins)	Delay Ratio
500	75	3.73
500	75	3.92

TABLE V: Bandwidth Attack Results for BitTornado (40 Attackers; 100 MB File; Leechers: 20 KB/sec & 40 KB/sec; Attackers: No Bandwidth Cap)

Table V shows the delay results for BitTornado bandwidth attack when the bandwidth settings correspond to the second scenario described in the fluid model ($b_A \geq u/n > b_L$). Even in these cases, the delay ratios observed are always less than 4. We observed during the experiments that, 3 out of the 4 unchoke slots at the seed are always occupied by attackers. and the remaining 1 slot (optimistic unchoke) is occupied either by an attacker or a leecher. However, since the leechers can occasionally get optimistic unchoke slots; and when they do, they can get a download bandwidth as high as 40 KB/sec, they can nevertheless download the entire file without much delay.

In summary, both Azureus and BitTornado seeds are quite resilient against bandwidth attacks.

V. Connection Attack against Initial Seeds

A. Attacking Azureus

We launched 800 attackers in various experiments for the connection attacks against Azureus seeds. All the attacks were very successful. Even for the cases with seed bandwidth as high as 140 KB/sec, the download at the leechers was completely stopped. Recall that for each of the experiments, 5 leechers were started at first, followed by all attackers, and then the remaining 25 leechers. In each experiment, the first 5 leechers (that are started at first) obtain the connection slots to the seed but the rest of the connection slots (45) are all occupied by attackers. We plot the bandwidth sharing of the seed bandwidth (140 KB/sec) between attackers and leechers in one typical connection attack experiment in Fig. 3. We observe that up to around 35 minutes from the start of the attack, leechers occupy most of the seed bandwidth while attackers receive only a small fraction. However, after 35 minutes, there is no seed bandwidth occupied by the leechers. At $t = 35$ minutes, all the initial 5 leechers have lost connections to the seed and all the 50 slots are occupied by the attackers. Even after $t = 35$ minutes, the attackers are still using very little bandwidth (roughly 200 kbps = 25 KB/sec). This is indeed a connection attack.

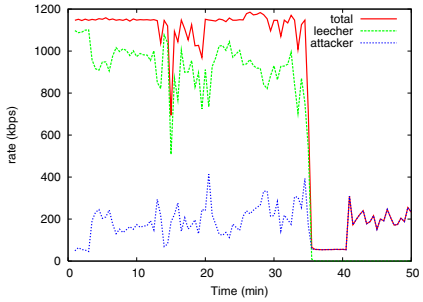

Fig. 3: Seed Bandwidth Distribution for Leechers and Attackers in an Azureus Connection Attack

In order to understand the connection attack results for Azureus, we further analyzed its source code and determined its connection management algorithm and state-transition procedure (Fig. 4). In Azureus, as soon as a connection request is received, it is accepted immediately if the total number of connection slots currently occupied has not reached the maximum limit c. Azureus also maintains a finite-size queue called "Discovered Peers" containing peers that have been discovered from the tracker, DHT, and gossiping (PEX). Every second, if there is a free connection slot, the seed initiates a connection to the peer in the front of the "discovered" list, performing an "Optimistic Connect".

Every 30 seconds, if all connection slots are full, the operation of "Optimistic Disconnect" is performed. The peer to which the seed sent data to furthest in the past (provided that data has been sent to the node at least once) is selected. If this value is greater than 5 minutes for the peer, the connection to this peer is closed by the seed.

Even if the attackers have relatively lower bandwidth compared to the leechers, the second unchoke criterion – fewer amount of data bytes downloaded so far – helps them grab some upload slots. Hence, the attackers can together prevent the leechers from obtaining slots for more than 5 minutes. One after another, the initial 5 leechers are disconnected from the seed as the operation result of "Optimistic Disconnect". Note that the attackers may also be optimistically disconnected.

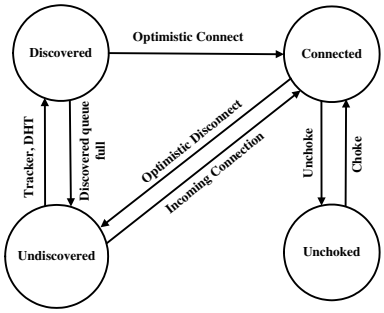

Fig. 4: State Diagram for Connection Management in Azureus

However, since they have been configured to continuously attempt to connect to the seed, they quickly reconnect to the seed after the previous disconnection, never giving the seed a chance to initiate connections to or accept connections from legitimate peers in the swarm.

B. Attacking BitTornado

Seed Uplink (KB/sec)	Avg Download Time w/o Attackers (mins)	Delay Ratio
50	177	2.52
50	177	2.99
100	176	1.00
100	176	1.00
150	167	1.03
150	167	1.03

TABLE VI: Connection Attack Results for BitTornado (800 Attackers; 500 MB File; Leechers: 48 KB/sec & 192 KB/sec; Attackers: 10 KB/sec & 20 KB/sec)

Table VI shows the download delay results for connection attacks on BitTornado seeds. The connection attack is again not very effective. This is a direct consequence of the bandwidth-first seeding algorithm used by BitTornado. For the high bandwidth seed, the bandwidth allocated per slot u/n is higher than the download rate of an attacker (20 KB/sec) but lower than that of the leechers. The 5 initial leechers that are connected to the seed always win over the attackers when fighting for unchoke slots. Hence, these 5 leechers can download the entire file quickly and redistribute it to the other 25 leechers.

Since both the bandwidth and connection attacks against BitTornado were not very successful, we also performed tests for combined bandwidth/connection attacks under the second scenario in the fluid model ($b_A \geq u/n > b_L$). Table VII shows the results. Delay ratio values of up to almost 9 were observed in these cases. In these test cases, we noticed that 3 out of the 4 slots at the seed were always occupied by attackers and the remaining 1 slot (optimistic unchoke slot) was shared among attackers and leechers. Due to the high attacker to leecher ratio ($9:1$) connected to the seed, the optimistic unchoke slot was also mostly occupied by an attacker. Thus, the delay ratio observed is more than double of that observed in the pure bandwidth attack case in the second scenario (Table V; note that all other parameter settings, except the number of attackers, are the same).

Seed Uplink (KB/sec)	Avg Download Time w/o Attackers (mins)	Delay Ratio
500	75	8.06
500	75	8.66

TABLE VII: Combined Bandwidth/Connection Attack Results for BitTornado (800 Attackers; 100 MB File; Leechers: 20 KB/sec & 40 KB/sec; Attackers: No Bandwidth Cap)

In summary, although the Azureus seed is vulnerable to connection attacks due to a (perhaps overly) sophisticated connection management algorithm, nevertheless, the vulnerability can be easily fixed. The BitTornado seed, using simpler and more conventional seeding and connection-management algorithms, is more resilient to connection attacks as well as combined bandwidth and connection attacks.

VI. Conclusion

In this paper, we investigated two natural attacks against initial seeds in BitTorrent: bandwidth attack and connection attack. To study these attacks, we chose three popular seeds – Azureus, uTorrent, and BitTornado. We developed a simple fluid model for the bandwidth attack, studied a few popular clients, and deployed an attack environment in PlanetLab. Our measurement results showed that bandwidth attacks are largely ineffective against Azureus and BitTornado seeds. We conclude that even when the attackers can detect a target torrent quickly enough and attempt to attack the initial seed, they are not able to "nip the torrent in its bud" rendering BitTorrent resilient to seed attacks.

Acknowledgements

This work has been in part supported by the NFS grant CNS-0917767. Xiaojun Hei's work was supported by the NSFC under Grant No. 60972014, the Fundamental Research Funds for the Central UniversitiesHUST: 2010MS109, the Technology Support Plan of the National 'Eleventh Five-Year-Plan' of China under Grant No. 2009BAH51B02 and the Technology Support Plan of the National 'Twelfth Five-Year-Plan' of China under Grant No. 2011BAK08B01. Di Wu's work was supported in part by NSFC (61003242), NSF of Guangdong Province(10451027501005630), the Fundamental Research Funds for the Central Universities(09LGPY56), the Doctoral Fund of Ministry of Education of China (20100171120047).

References

[1] P. Dhungel, X. Hei, D. Wu, and K. W. Ross, "The seed attack: Can BitTorrent be nipped in the bud?" Tech. Rep., 2009. [Online]. Available: http://cis.poly.edu/~ross/papers/SeedAttack.pdf

[2] B. Fan, J. C. S. Lui, and D.-M. Chiu, "The design trade-offs of bittorrent-like file sharing protocols," *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, pp. 365–376, April 2009.

[3] K. El Defrawy, M. Gjoka, and A. Markopoulou, "BotTorrent: Misusing BitTorrent to launch DDoS attacks," in *Proc. USENIX SRUTI*, Santa Clara, CA, June 2007.

[4] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang, "Exploiting BitTorrent for fun (but not profit)," in *Proc. IPTPS*, Santa Barbara, CA, Feb 2006.

[5] M. A. Konrath, M. P. Barcellos, and R. B. Mansilha, "Attacking a swarm with a band of liars: evaluating the impact of attacks on BitTorrent," in *Proc. IEEE P2P*, 2007.

[6] P. Dhungel et al., "A measurement study of attacks on BitTorrent leechers," in *Proc. of IPTPS*, Tampa Bay, Florida, Feb 2008.