Is BitTorrent Unstoppable?

Prithula Dhungel†, Di Wu†, Xiaojun Hei‡, Brad Schonhorst†, Keith W. Ross†
Polytechnic University, Brooklyn, NY 11201†
Hong Kong University of Science and Technology, Hong Kong‡

ABSTRACT

Anti-P2P companies have begun to launch Internet attacks against BitTorrent swarms. We use passive and active Internet measurements to study how successful these attacks are at curtailing the distribution of targeted content. For our active measurements, we develop a crawler that contacts all the peers in any given torrent, determines whether leechers in the torrent are under attack, and identifies the attack peers in the torrent. We use the crawler to investigate 8 top box-office movies. Using passive measurements, we perform a detailed analysis of a recent album that is under attack and evaluate the effectiveness of attacks on leechers. Using private torrents created in Planet Lab, we investigate attacks against the initial seed in the early stages of a torrent. For some of the more interesting attacks, we also develop simple analytical models, giving additional insights into the potential vulnerabilities of BitTorrent. Finally, we address whether the BitTorrent architecture is fundamentally more resilient than earlier file sharing architectures (e.g., Kazaa and eDonkey) against Internet attacks.

1. 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. These attempts include numerous law suits against P2P file sharing companies (against Napster, Kazaa and many others), tracking and suing users of P2P file sharing systems [19], and most remarkably, launching large-scale Internet attacks against the P2P systems themselves. The large-scale Internet attacks have been performed by specialized anti-P2P companies, also known as "interdiction" companies (e.g., Media Defender [8], Safenet [13] and Marcovision [7]), working on the behalf of the RIAA and specific record labels. Several studies showed that these attacks were successful at severely impeding the distribution of targeted content over several P2P file sharing systems, including FastTrack/Kazaa, Overnet/eDonkey, and Gnutella /Limewire [27, 20, 28]. In particular, these attacks, along with the law suits, contributed to the demise of the Kazaa/FastTrack file-sharing system.

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 [31]. Unlike Napster and Kazaa, BitTorrent uses an open protocol, which has been implemented by more than 50 different clients [3]. The BitTorrent client is thus not developed and distributed by a single company, which can be easily targeted for a lawsuit. Also included in the BitTorrent ecosystem are torrent discovery and peer discovery services, which can potentially be legally attacked; in fact, in late 2004, Suprnova, the largest torrent discovery site at that time, was closed after legal threats. Today, however, there are many torrent discovery and peer discovery services scattered around the globe, all of which are defying legal threats. Moreover, peer discovery can be decentralized using DHTs and gossiping, as is currently being done with clients such as Azureus and uTorrent.

Given that it is currently difficult, if not impossible, to stop BitTorrent by suing companies, and suing individual users is both painstaking and unpopular, the only remaining way to stop BitTorrent is via Internet attacks. Not surprisingly, the music, film, and television industries have begun to hire anti-P2P companies to curtail the distribution of "assets" in BitTorrent [17, 5, 6].

It is important to understand just how vulnerable the BitTorrent ecosystem is to large-scale, resource-intensive attacks for several reasons. First, BitTorrent has proven to be extremely effective at distributing large files, including open-source software distributions. According to Evidenzia [9], the number of files distributed on BitTorrent almost tripled from 2006 to 2007, and that visits to major torrent location sites PirateBay and Mininova doubled over this same period [4]. Given BitTorrent's large footprint in the Internet, it is of interest to know whether it is vulnerable to attack and – if successfully attacked – whether its impact on the Internet will be consequently reduced. Second, having proven itself capable of massive-scale file distribution, BitTorrent now serves as a blueprint for other P2P systems, including

on-demand and live streaming applications [11, 12, 14]. Understanding BitTorrent's vulnerabilities will therefore shed insight into the vulnerabilities of a broader class of P2P systems. Third, the music and film industries are clearly interested in whether network attacks can successfully curtail file distribution over BitTorrent. If not, they can abandon financing the attacks and investigate other solutions and business models.

Without taking sides on issues of copyright and fair use, in this paper we investigate just how vulnerable the BitTorrent ecosystem is to large-scale, resource-intensive attacks. In particular, by attacking one or more components of the BitTorrent ecosystem, is it possible to stop or curtail the distribution of a targeted file? To answer these questions, we undertake a number of complementary measurement studies:

- We developed a crawler that contacts all the peers in any given torrent, determines whether the leechers in the torrent are under attack, and identifies the attack peers. We used the crawler to analyze 8 current top box-office movies.
- Using passive measurements, we performed a detailed analysis of a recent album that is under attack. For these measurements, we developed a customized packet parser, which identifies the attack peers and the type of attack they employ. We also employ passive measurement to evaluate the effectiveness of ongoing anti-P2P attacks against BitTorrent leechers.
- We created our own private torrents using peers dispersed in Planet Lab (PL) as well a small number of university and residential hosts that are fully under our control. We limit the upload and download rates of the PL nodes to mimic BitTorrent torrents in the wild. Leveraging this measurement infrastructure, we investigate the impact of attacks on BitTorrent seeds.

In addition to the measurement studies, when appropriate, we provide simple analytical models for attacks and defenses, providing further insights into BitTorrent vulnerabilities. We also explore whether the current BitTorrent architecture is fundamentally more resilient to Internet attacks than earlier file sharing systems (such as Kazaa and eDonkey).

This paper is organized as follows. We conclude this section we a brief discussion of related work. In Section 2, we give a summary of the modern BitTorrent ecosystem, focusing on the characteristics that are particularly relevant to our vulnerability study. In Section 3, we provide a taxonomy of BitTorrent attacks, covering attacks on seeds, leechers, peer discovery, and torrent discovery. In Sections 4 and 5, we use passive and active measurement methodologies to study ongoing attacks against leechers. In Section 6, we create a Planet Lab environment to analyze attacks against seeds. Finally, in Section 7, we draw conclusions about BitTorrent vulnerability based on our taxonomy, measurements, and analytical results.

1.1 Related work

Recently, there has been a number of studies on pollution and poisoning attacks on second-generation P2P file sharing systems (such as Kazaa and eDonkey). Analytical and simulation studies showed that the pollution attack (inserting corrupted content into the P2P file sharing system) is not scalable in co-operative P2P networks and that these attacks were successful in some networks only because of the user behavior [23]. On the other hand, a measurement study concluded more than 50% of the files in Kazaa to have been polluted [27]. Similarly, the "index poisoning", wherein an attacker advertises an enormous number of bogus sources for a targeted content, was highly pervasive in the FastTrack and Overnet DHT (eDonkey) networks [28].

To date there has been little work on attacks on BitTorrent that aim at curtailing the distribution of targeted files. A recent simulation analysis of BitTorrent examined two attacks: an attack involving tampered buffer maps, and a connection monopolization attack [25]. The simulation analysis assumed idealized BitTorrent clients using an idealized protocol. (In practice, developers of BitTorrent clients deviate significantly from the textbook BitTorrent protocol.) To gain meaningful insights into BitTorrent vulnerabilities, measurement studies with real-world BitTorrent clients are required. This current submission extends an earlier workshop paper [22], which focused on attacks against BitTorrent leechers. The current paper provides a measurement study of both seed and leecher attacks, a taxonomy of BitTorrent attacks, and insightful analytical models of attacks. To the best of our knowledge, this submission is the only measurement study of BitTorrent attacks to date.

There has been a number of studies on how P2P systems can be used as engines for DDoS attacks against arbitrary hosts in the Internet [29, 33, 18]. There has also has been a number of studies on whether BitTorrent's tit-for-tat incentive mechanism is sufficient for preventing freeriders from downloading files [24, 30, 32]. These two classes of attacks (DDos attacks from P2P systems and free rider attacks) are orthogonal to the attacks considered in this paper.

2. BITTORRENT ECOSYSTEM TODAY

Over the years, the BitTorrent ecosystem has evolved and become rather complex, with several recent developments that are of particular interest in this study. For our purposes, the BitTorrent ecosystem consists of

four major components: seeds, leechers, peer discovery, and torrent discovery. Each of these four components can potentially be attacked.

A *torrent* is the collection of peers that participate in the distribution of a specific file. At any given instant of time, each peer in a torrent is either a *leecher* or a *seed*. A seed possesses the entire file, uploads content to leechers, but does not download since it already has the file. A *leecher* uploads content to other leechers and downloads content from seeds and leechers. Typically a torrent begins with an *initial seed,* which is the only peer to have the file.

Each leecher and seed uses one of the many *BitTorrent clients*. Some of the more popular clients today include Azureus[1], uTorrent [16], and BitComet [2]. More than 50 BitTorrent clients have been developed to date [3]. BitTorrent clients communicate with each other using the BitTorrent protocol. However, some clients (e.g., Azureus) have started to use their own customized protocols. When two Azureus clients communicate, they use the Azureus protocol; when an Azureus client communicates with a non-Azureus client, they use the conventional BitTorrent protocol. We stress that the BitTorrent clients are developed by independent developers with a significant amount of freedom and autonomy. In particular, developers can create their own seeding algorithms (as discussed subsequently), can use piece-selection algorithms that are different from rarest first, and so on. We have observed that many clients deviate significantly from Bram Cohen's Mainline client [21]. For this reason, in drawing conclusions about BitTorrent behavior and vulnerabilites, it is important to use popular real-world clients, rather than simulations of idealized clients.

One common technique for *peer discovery* is to use a *tracker*: each peer in the torrent registers with the tracker, and any peer can contact the tracker at any time to obtain a random subset (IP-port pairs) of other peers in the torrent. Communication between a peer and a tracker is over HTTP. Today, many BitTorrent clients (e.g., Azureus, uTorrent) additionally provide peer discovery using DHTs. For example, all active Azureus clients (that elect to participate in the DHT) communicate with each other (across all torrents) via the DHT. An Azureus client can then query the DHT, using a torrent identifier for the key, to obtain a list of Azureus peers participating in the torrent. Thus an Azureus client typically discovers peers in a torrent both from the torrent tracker and from other active Azureus clients. Furthermore, many BitTorrent clients (including Azureus and uTorrent) now allow peers in a torrent to exchange peer lists directly with each other.

Users learn about the existence of ongoing torrents from *torrent discovery sites* such as PirateBay, Mininova, Isohunt, Torrentspy, Legaltorrent, snarf-it, and BiteNova [15]. These sites typically provide a search interface. In response to a user query, the search returns a list of torrents that match the query as well corresponding torrent description information such as the name of the file, file size and type, the date the file was originally seeded, the current number of seeds and leechers, and often comments from users about the usability and quality of the file. For each of these results, the discovery site has a copy of a .torrrent file, which includes the IP addresses of one or more trackers and the hashes of all the pieces in the file. To download a file, the user obtains the .torrent file, contacts the tracker in the .torrent file, and joins the torrent. We remark that some of these torrent discovery sites provide a tracker service (e.g., PirateBay), whereas other sites (e.g., torrentz.com) provide no tracker servers and instead crawl torrent discovery sites and index the totality of the discovered torrents.

When a user wants to start a new torrent, it needs to seed the file and register the torrent with a tracker. To this end, the user may use its own tracker or it may use a tracker service that is provided by some of the torrent discovery sites (e.g., PirateBay). For example, a user can simply create a seed by starting a BitTorrent client, including the file it wants to distribute in its shared folder, and registering the torrent with a tracker. The original BitTorrent protocol specification used the *Bandwidth First* seeding algorithm that favored neighboring peers with higher download rate when uploading blocks [26]. Recently, additional algorithms have been specified. In the *Round Robin* algorithm, the seed allocates equal time slots to all neighboring peers; each peer gets its share of bandwidth from the seed at regular intervals of time. We have observed that the popular BitTorrent clients today often use a combination of these two algorithms, and that the actual seeding algorithm employed often varies from one version to the next of the same client.

3. BITTORRENT ATTACKS: TAXONOMY AND ANALYSIS

If a company or an individual wants to thwart the distribution of specific content (e.g., a movie, television series, or record album) in BitTorrent, it can take one of two broad approaches: attacking an existing torrent in the BitTorrent system; or attacking by creating decoy torrents. In this section we present a taxonomy of attacks and also present simple analytical models for some of the attacks. In the subsequent sections, we carry out measurement studies of some of the more potentially damaging attacks. Our focus in this paper is on attacking existing torrents; however, in presenting our taxonomy, we also briefly discuss decoying torrents in this section. We emphasize that it is not possible to identify every possible BitTorrent attack that could

arise someday; instead we mostly focus on attacks that are actually being deployed by anti-P2P companies.

3.1 Attacking an Existing Torrent

The BitTorrent system consists of four major components: leechers, seeds, peer discovery, and torrent discovery. Each of these components can potentially be attacked in order to curtail the distribution of specific content.

3.1.1 Attacking the Leechers

In this class of attacks, the attacker targets the individual leechers in the torrent, with the goal of preventing the leechers from obtaining the entire file, or at least substantially prolonging the download time in order to frustrate the user. One obvious approach is to launch bandwidth flooding DoS attacks against all the leechers in the torrent. For a large torrent, this could require exorbitant bandwidth resources for an extended period of time (since new leechers are continually joining the torrent). Moreover, with bandwidth flooding, ISPs may either filter the DoS traffic and/or file complaints. A more feasible attack against a leecher would instead exploit specific characteristics of the BitTorrent protocol, which can be viewed as having two layers: a *connection layer* and a *piece layer*. We now describe how the protocol can be exploited in each of these two layers.

Connection Attack

A BitTorrent peer limits the number of simultaneous TCP connections it has with other neighboring peers. This limit is client and configuration specific, but is typically on the order of 50 to 100. The goal of the connection attack is to try to tie up as many of these connection slots as possible, if not all of them. To this end, for each target leecher, the attacker attempts to establish a large number of TCP connections and maintain these connections as long as possible. Over these TCP connections, the attacker uploads little if any data blocks, and therefore does not require to have excessive bandwidth resources. To maintain the connections, depending on the specific BitTorrent client employed by the leecher, the attacker may have to periodically exchange BitTorrent messages with the leecher.

To succeed at this attack, the attacker must be able to establish a large number of connections to the target leecher (or have the leecher establish a large number of connections to attacker hosts via a complementary peer-discovery attack), and succeed at maintaining these connections for extended periods of time. To defend against this attack, a client can be designed to drop neighbors that do not provide it blocks (either because it does not respond to a block a request or because it never has any needed pieces). Also, blacklists for suspected attack IPs (or suspected /24 networks) can be created and deployed.

Currently anti-P2P companies are launching connection attacks against specific client types in targeted torrents. In the subsequent section, we will present both passive and active measurement results for this attack.

Piece Attack

Recall that in BitTorrent, each file is divided into *pieces*, where each piece is typically 256 KBytes. Each piece is further divided into *blocks*, with typically 16 blocks in a piece. When downloading a piece, a client requests different blocks for the piece from different peers. After obtaining all the blocks in a piece, the client reassembles the piece, calculates a SHA1 hash, and compares the result with the corresponding hash value in the .torrent file. If there is a hash failure, that client deletes the entire piece and re-downloads each of the piece's blocks (potentially from different peers).

In the *piece attack*, the goal of the attacker is to have the hash check frequently fail at the targeted leecher. This would cause the leecher to waste time and bandwidth resources in acquiring the file, thereby prolonging the download time of the file. Ideally, the attacker would like to create a large number of hash failures by using as little attack bandwidth as possible. To meet these goals, the attacker joins the torrent by registering itself to the corresponding tracker. After becoming a neighbor of a victim leecher, it advertises that it has a large number of pieces of the file. Upon receiving this information, the victim leecher peer sends a request to the attack peer for a block. Instead of sending the authentic block, the attacker sends a fake one. After downloading all the blocks in the piece (from the attack peer and from other benevolent peers), the hash check then fails due to the fake block from the attacker. This requires the victim peer to download the entire piece (16 blocks) again, delaying the download of the file. If the peer chooses to download any of the blocks again from this or another piece attacker, the download is further delayed. Note that an attacker can cause a victim peer to waste 256 KBytes of download bandwidth by only sending it a 16 KByte block (using typical numbers).

Currently anti-P2P companies are launching piece attacks against specific client types in targeted torrents. In the subsequent section, we will present both passive and active measurement results for this attack. We will also describe an adaptive defense mechanism for this attack.

To gain some preliminary insight into the piece attack, we now provide a simple, back-of-the-envelope analytical model of the attack. Consider a peer that wants to obtain a specific piece consisting of 16 blocks. Suppose this peer currently has n neighboring peers that claim to have the piece. Of these n neighbors, let m

denote the number of attack peers. Let k denote the number of neighbors the peer contacts to obtain the 16 blocks (a typical value of k is 5). A hash failure occurs if one or more of the k chosen neighbors is an attack peer. Assuming the k download neighbors are chosen from the n neighbors at random, the probability that a clean piece is downloaded is:

$$\begin{aligned} p &= P(\text{download a clean piece}) \\ &= \binom{n-m}{k} / \binom{n}{k} \\ &= \frac{(n-m)(n-m-1)\ldots(n-m-k+1)}{n(n-1)\ldots(n-k+1)} \\ &\approx (1-\frac{m}{n})^k \end{aligned}$$

where the last approximation holds when $k \ll n$ and $k \ll n-m$. Let γ denote the fraction of attack peers among the n neighbors with the piece, that is, $\gamma = m/n$. Figure 1 shows the probability of downloading a clean piece as a function of γ for $k = 5$. Note that the probability of downloading a clean piece decreases rapidly as γ increases.

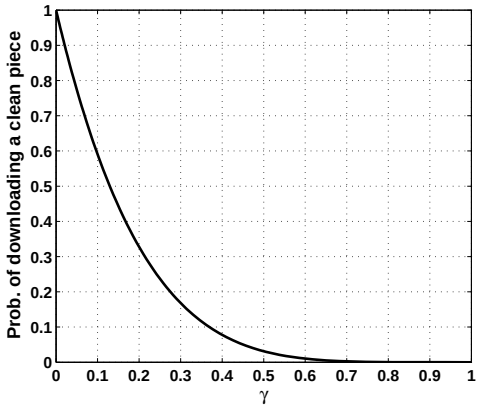


Figure 1: Probability of downloading a clean piece vs. fraction of attack peers.

From Figure 1, we see that the attackers need a high value of γ to be effective. For example, if they want a failure probability of 70%, approximately 20% of the neighboring peers should be attack peers. Note that if a peer is attempting to download a rare piece, for which relatively few benevolent peers have a copy, γ will increase. Similarly, during the end game, when a peer seeks a small number of missing pieces, there may be relatively few benevolent peers that have the missing pieces, also causing γ to increase. In fact, we have observed (Section 4) that the piece attack is most effective in the end game.

3.1.2 Attacking the Seeds

In this class of attacks, the attacker attempts to diminish the seeds' ability to upload blocks (and ideally stop the seeds from uploading altogether). This approach has the potential to be particularly effective in the early stages of a torrent, when there is only the initial seed and the leechers collectively have not obtained the entire file. 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.

One approach is to bandwidth flood the seed's downstream bandwidth, preventing legitimate peers from even connecting to the seed. Because the number of seeds is often small, particularly in the early stages of a torrent, brute-force bandwidth flooding against seeds may be feasible for some torrents, particularly if the seed has a relatively low-speed Internet connection. A DDoS bandwidth flooding attack can be used to avoid detection by ISP filtering mechanisms. The main defense (as is typically the case for bandwidth DoS attacks) is to increase the seed's downstream bandwidth.

We now describe attacks that are specific to BitTorrent and, in particular, to BitTorrent's seeding algorithms.

Bandwidth Attack

Recall that BitTorrent seeds usually employ bandwidth-first or round-robin algorithm or some combination of the two. Consider now attacking a seed that employs bandwidth first. The seed will have connections with a large number of peers (typically 50-100), but only uploads to the K neighbors that download at the highest rate, with a typical value of K being 4-7. If the attacker can make K connections to the seed (from one or more attack peers) and become the K highest downloaders, then the attack peer can significantly diminish the seed's capability of sending blocks to legitimate peers. (Because the seed will optimistically unchoke other peers, some blocks will still be sent to legitimate peers.) To be successful, the attack peers may need to collectively download at a rate close to the seed's upstream bandwidth. However, because many seeds are behind asymmetric access links, significantly less bandwidth resources may be needed than for flooding the downstream bandwidth of the seed.

To gain some preliminary insight into this bandwidth attack, we now provide a simple analytical model of the attack. Suppose there are M attack peers and N legitimate peers connected to the seed. For simplicity, we assume that all the attack peers have the same downstream bandwidth of x and all the legitimate peers also have the same downstream bandwidth of y, and $x > y$. Suppose the seed has upstream bandwidth of U, and provides K simultaneous upload slots. Among the K upload slots, $K-1$ slots are for regular unchoked peers, and 1 slot is for optimistic unchoked peer. For an unchoked peer, its download rate is bounded by the mini-

mum of its download bandwidth and $\frac{U}{K}$. Let us consider two scenarios:

- $x > y \geq \frac{U}{K}$: when competing for the seed bandwidth, attack peers have no advantage over legitimate peers under the Bandwidth First seeding algorithm. The fraction of seed upstream bandwidth obtained by attackers is determined by the percentage of attackers among connected peers, which equals $\frac{M}{N+M}$.
- $x \geq \frac{U}{K} > y$: In this case, a legitimate peer's download rate is less than U/K. Thus, under the Bandwidth First seeding algorithm, all the regular unchoked slots are taken by attack peers. Legitimate peers can only get unchoked by optimistic unchoking. The fraction of seed upstream bandwidth obtained by attack peers is approximately given by $\frac{K-1}{K}$.

Connection-Monopolization Attack

Now consider attacking a seed that employs the round-robin uploading algorithm. Since each peer gets (in principle) an equal share of the bandwidth, the goal of the attacker is to fill up as many as possible of the connection slots of the seed so that few or no slots are left for the legitimate peers. To make sure that the seed maintains the connections, the attacker peers may need to occasionally download some blocks.

If at the start of the attack, the seed already has many ongoing connections to legitimate peers, the attacker will not be able to establish many of its own connections with the seed, and therefore may not significantly diminish the seed's upload capacity. When a legitimate peer leaves, the aggressive attacker can try to immediately take its place. However, if the seed has already distributed a large number of chunks to genuine peers before the attackers finally succeeds at monopolizing the connection slots, the attack might have little impact.

3.1.3 Attacking Peer Discovery

The goal of peer discovery attacks is to prevent the leechers from obtaining the IP-port pairs of the legitimate peers in the torrent. One such attack is to bandwidth flood the tracker. However, because tracker services often have high-bandwidth connections, bandwidth flood attacks are difficult. Furthermore, since many popular BitTorrent clients (including Azureus and uTorrent) enhance peer discovery with a DHT and/or gossiping, knocking out the tracker may not suffice.

Another approach is to poison the peer discovery mechanisms with non-authentic IP-port pairs. For example, if the attacker succeeds at poisoning 99% of the IP-port pairs in a tracker, then the vast majority of the random IP-port pairs the tracker gives to a leecher will be useless. For eDonkey/Overnet, anti-P2P companies also used an IP poisoning attack, in which they poisoned the index with a massive number of bogus IP addresses [28]. This was a successful attack, as finding a useful IP address was like finding a needle in a haystack. Poisoning was relatively easy in the Overnet DHT, as an arbitrary user can insert an arbitrary IP address (or file hash) into the DHT using UDP. It is more difficult, however, to poison a BitTorrent tracker, since attacker must establish TCP connections and send register messages at regular intervals just to maintain one poisoned IP. To maintain a large number of bogus IPs, the attacker needs to maintain a large number of TCP connections to the tracker. Moreover, because of TCP's three-way handshake, the attacker reveals its IP address, which (once identified as part of an attack) can be put on a blacklist.

3.1.4 Attacking Torrent Discovery

Finally, it is possible to attack the torrent index sites (such as PirateBay and Mininova). One possible attack is a bandwidth-flooding DoS attack. There has actually been a few such attacks documented. For example, in December 2004, some BitTorrent web servers, including that maintained by Lokitorrent, faced a DDoS attack for hours [5]. Another approach is to enter negative comments (e.g., "file does not play") about the targeted torrents on the Web sites.

3.2 Decoy Torrent Attacks

In this class of attacks, the attackers create their own "decoy" torrents for the targeted content. Specifically, the attacker seeds fake files (also called polluted files) that are tagged with the name of targeted content. For example, an attacker can upload one or more .torrent files for a targeted movie into a number of torrent discovery sites (PirateBay, Mininova and so on). Each of these torrent files advertises the name of the targeted movie. However, for each of the torrents, instead of seeding the actual movie, the attacker seeds one or more "fake" files for the movie. Another version of the decoy attack, is to have the decoy seed distribute only 99% of the file, with the file being either the actual movie or some bogus file. In this case, the leechers remain waiting indefinitely for the last few pieces to complete the download until the user gets frustrated and abandons the download altogether.

In both of the above techniques, the goal of the attacker is to make the number of decoys for the content larger than the number of authentic torrents for the movie, so that users download the decoy rather than an authentic version with a high probability. To increase the chances that users select decoy torrents, the attackers can deploy their own trackers (rather than use

the public trackers of torrent Web sites), which are directly controlled by the attackers. When the torrent Web sites contact these customized trackers, the trackers report false statistics, indicating a very large torrent size, which are in turn presented to the users. Users tend to choose large torrents, since large torrents are generally more trusted and provide the file faster.

As a defense to this attack, users typically post comments about torrents at the various torrent web sites; if a torrent is a decoy, it is easily flagged in these comments. Furthermore, users often inform the operators of these sites when a torrent is determined to be a decoy. Operators then remove the decoy. Many torrent discovery sites (e.g., PirateBay) also disallow the use of third-party trackers.

Leaked emails from one of the anti-P2P companies indicate that it has been using the decoy attack [6]. However, we have not observed widespread use of this attack in BitTorrent. Therefore, in this paper, we focus on attacks launched against existing torrents and do not further consider decoy-torrent attacks.

4. BITTORRENT ATTACKS ON LEECHERS - PASSIVE MEASUREMENT

Having set the stage with a taxonomy of attacks, we now measure the effectiveness of several attacks, many of which are either actually being deployed. We use passive and active measurements of torrents in the wild, as well as private torrents running on PlanetLab. Throughout this measurement study, we focus on two very popular clients, Azureus and uTorrent. In this section, we begin with passive measurements to evaluate the effectiveness of connection attacks and piece attacks against leechers.

4.1 Passive Measurement Methodology

While repeatedly downloading a file suspected to be under attack, we collected multiple packet traces from hosts connected to both Ethernet and DSL access networks. On each host, we captured all the incoming and outgoing packets. We also developed our own packet parser to identify different types of attackers in the trace and analyze their behaviors.

To measure the performance of BitTorrent without attacks, we used a third-party software, PeerGuardian [10], to prevent connections to and from the IP ranges in a specified blacklist. Our IP blacklist is based on the ZipTorrent blacklist published on *torrentfreak.com* [17]. Note that, since the anti-P2P companies (e.g., MediaDefender [8]) change the IP range of their attack hosts, this blacklist is not always complete and may not always eliminate all the attacker hosts.

4.2 Passive Measurement Results

In this section we present measurement results for a torrent for the new album titled *"Echoes, Silence, Patience & Grace"* from *"Foo Fighters"*, which we suspected to be under attack. This popular album was released on September 25, 2007, a few weeks before our experiments. At the time of the experiment, it held the number 1 position on the UK album chart and iTunes ranking list. The size of the file is 108 MBytes. In our testing, we downloaded the file from this torrent 54 times.

4.2.1 Azureus Client

Because Azureus clients can import IP blacklist, we use this Azureus feature to perform IP filtering. Within one day, we performed downloads for this torrent multiple times using Azureus clients, and switched the IP filter on or off alternatively. First we present the basic average download-time statistics in Table 1.

Azureus	w/ IP-filtering	w/o IP-filtering	DR
Ethernet	15.52 mins (6 downloads)	20.99 mins (6 downloads)	35.2%
DSL	19.98 mins (6 downloads)	25.88 mins (6 downloads)	29.5%

Table 1: Average downloading time using Azureus clients

In Table 1, *Delay Ratio (DR)* is defined as follows to evaluate the effectiveness of attacks in lengthening downloading time,

$$\text{Delay Ratio} = \frac{T_d \text{ w/o IP-filtering} - T_d \text{ w/ IP-filtering}}{T_d \text{ w/ IP-filtering}}$$

where T_d is the average downloading time of BitTorrent clients. From the table, we clearly observe that the downloading time of the file is prolonged when attacked. For both DSL and Ethernet peers, the download time on average increased by about 30%. The actual increase in download time may be larger, since we may not have blacklisted all the malicious peers. However, given the download rate of the DSL client, the size of the file, and that the minimum observed download time was 17 minutes, it is unlikely that the average download time without an attack would have been less than 17 minutes. Thus, we can safely say, at least for DSL, that the attackers did not prolong the downloading of this file by more than 50%.

To get a deeper understanding of the attack on Azureus clients, we selected one typical packet trace and analyzed it with the packet parser we developed. Our parser can categorize all the IPs in the trace into different types as follows:

- *No-TCP-connection Peers:* peers with which our client cannot establish TCP connections.

- *No-BT-handshake Peers:* peers with which our client can successfully establish TCP connection, but when the client sends a BitTorrent handshake message, the peer does not return a BitTorrent handshake response.
- *Connection-Attack Peers:* peers that occupy the connection slots of our client and simply chat with repeated BitTorrent protocol messages, without uploading any data. For Azureus clients, we consider any peer that sends more than one Azureus handshake message as a *Connection-Attack Peer*.
- *Piece-Attack Peers:* peers that upload fake blocks to our client. To identify piece-attack peers, we first need to check whether hash fails happened during downloading. When a hash fails, we identify all the IPs that have uploaded blocks for the piece and check whether the uploaded blocks are fake or not.
- *Benevolent Peers:* peers that communicate normally with our client via the BitTorrent protocol and upload at least one genuine block to our client.
- *Other Peers:* peers that don't fall into any of the above categories.

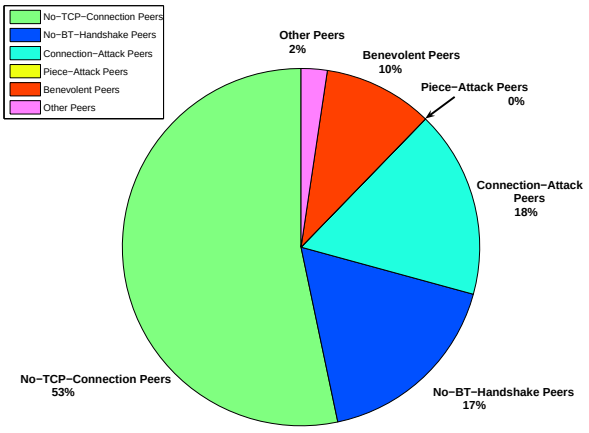


Figure 2: Peer distribution in Azureus trace

Figure 2 shows the distribution of different types of peers in the Azureus trace. Among all the IPs our Azureus client attempted initiating a connection to, over half of them could not be reached. The high percentage of no-TCP-connection peers is not necessarily due to attackers. The no-TCP-connection peers include NATed peers, firewalled peers, stale IPs returned by trackers or gossiping messages, and peers that have reached their limit on TCP connections (typically around 50-80 in BitTorrent). Even in clean torrents (e.g., public-domain software) where no attacks exist, we observed a large percentage of no-TCP-connection peers.

No-BT-handshake peers account for 17% of the total IPs. If combined with no-TCP-connection peers, almost 70% of all the IPs are not useful for our Azureus client. For the remaining 30% of the IPs, only 10% of the IPs are benevolent peers, while 18% IPs belong to connection-attack peers, which chat with the Azureus client continuously but without uploading any piece. Connection-attack peers account for a majority of useful peers (i.e., 60%).

To estimate how chatty the attackers actually are, we checked the number of handshake messages sent out by each connection-attack peer (Figure 3). We can observe that most of connection-attack peers are very chatty, and send out as many as 40-60 handshake messages to our Azureus client. Those connection-attack peers persist as neighbors of the Azureus client during the downloading process, and hinder the client from contacting benevolent peers. No hash fails occurred during

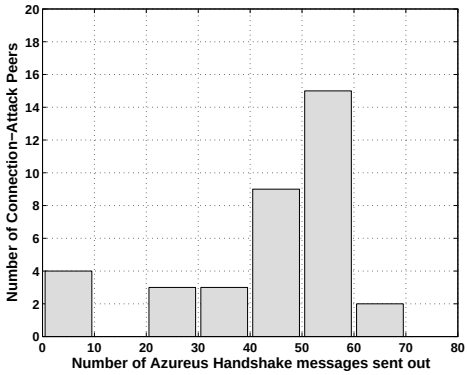


Figure 3: Distribution of Azureus handshake messages across connection-attack peers.

the downloading. Thus, it appears that the attackers did not launch a piece attack against Azureus clients at this time.

4.2.2 *uTorrent Client*

We also used uTorrent clients to download the same file. We turned off the automatic filtering function of uTorrent and used PeerGuardian to perform IP-filtering.

uTorrent	w/ IP-filtering	w/o IP-filtering	DR
Ethernet	9.17 mins (10 downloads)	9.42 mins (10 downloads)	2.7%
DSL	18.32 mins (5 downloads)	28.93 mins (5 downloads)	57.9%

Table 2: Average downloading time for uTorrent clients

Table 2 provides the average downloading time of uTorrent clients. For uTorrent clients with Ethernet connections, the attackers did not succeed at significantly increasing the average download time. However, the attackers appear to have some success with DSL clients, increasing the average download time by 58%.

uTorrent	w/ IP-filtering	w/o IP-filtering
Ethernet	1.7 Hash Fails	44.2 Hash Fails
DSL	4.2 Hash Fails	68.4 Hash Fails

Table 3: Average number of Hash Fails for uTorrent clients

Table 3 shows the average number of hash fails for uTorrent clients. Compared with Azureus clients (which had no hash failures), hash failures occur much more frequently. The hash failures are a direct consequence of the piece attack being launched against uTorrent. Hash-failures may not significantly impact an Ethernet peer, since if the Ethernet peer can find one other high-bandwidth benevolent trading partner, it may be able to rapidly download from it complete pieces (all 16 blocks) even if the other neighbors are producing hash failures. For DSL clients, because of the tit-for-tat algorithm, the client is typically trading only with other lower-bandwidth peers; even if one of these peers is producing a stream of clean pieces, the pieces would be coming in at a relatively low rate.

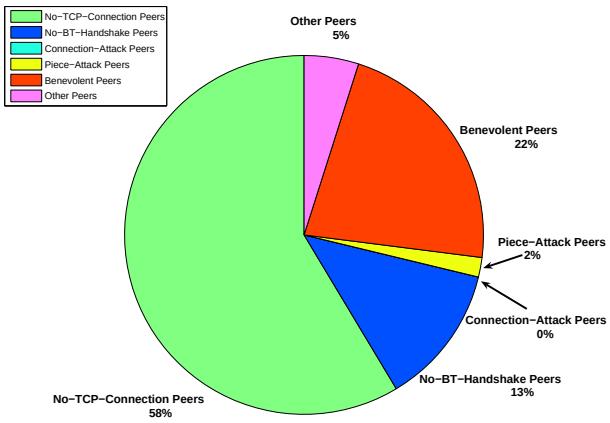


Figure 4: Peer distribution in uTorrent trace

To gain deeper insights, we plot the peer distribution in one of uTorrent traces in Figure 4. Similar to the Azureus trace, no-TCP-connection peers account for 58% of all the IP addresses in the peer list.

Compared with the Azureus trace, the main difference lies in the distribution of connection-attack peers and piece-attack peers. In Azureus, we saw significant connection-attack activity but no piece attacker. In case of Azureus, the attackers exploited the implementation vulnerability of not being able to detect malicious behavior of attackers sending multiple handshake messages. It appears that uTorrent clients do not have this vulnerability.

However, we did observe the piece attack in uTorrent. The piece attack is different from connection attack in that it doesn't require many IP addresses to launch the attack. Even if the percentage of piece-attack peers is fairly low among all the IPs, the attack can still be effective, particularly towards the end of the file download (the end game).

In summary, the anti-P2P companies are applying distinctly different strategies against different BitTorrent clients. From this experiment (involving 54 downloads from the same torrent), we observe that the attacks are not always successful at significantly prolonging download times. For Ethernet clients, the attackers appear to be largely ineffective. On the other hand, average download times for our uTorrent client behind DSL connections increased by about 60%. However, even if download times were to double (100% delay ratio), it is not clear that many users would abandon BitTorrent, since users often download in the background or over night. In the next section, we examine the attacks over a wider array of torrents.

5. BITTORRENT ATTACKS ON LEECHERS - ACTIVE MEASUREMENT

In this section, we provide active measurement results for the detection of connection-attack peers and piece-attack peers in torrents for 8 top box office movies.

5.1 Active Measurement Methodology

We developed a crawler that traverses the BitTorrent network gathering IP addresses of peers for a given torrent. We also developed customized BitTorrent clients, and devised heuristics for the detection of connection-attack peers and piece-attack peers. Doing this enabled us to quickly run experiments over a large number of torrents without having to download the entire files (as in the previous section).

5.1.1 Crawler Architecture

The output of the crawler is a "pool" containing the IP address and port pairs of peers in the torrent. It repeatedly requests the tracker for lists of peers participating in the torrent. Every time a list is received from the tracker, the crawler checks each IP and port to see if it already exists in the pool. If not, the new pair is added to the end of the pool. After gathering some IP addresses and ports in the pool, an IP address and port

Table 4: Measurement Results for Connection-Attack Peers

Movie ID	Total Peers Crawled		IPs from Blacklist		Non-Useful Peers		Useful Peers	Connection-Attack Peers		
	Tracker	Gossip	Tracker	Gossip	Tracker	Gossip		Tracker	Gossip	IPs from Blacklist
Movie1	116	864	1	73	90	836	54	0	27	26
Movie2	633	206	1	48	528	159	152	0	7	7
Movie3	144	158	0	30	111	98	93	0	0	0
Movie4	16	407	0	12	8	398	17	0	2	0
Movie5	29	1460	0	2	16	1460	13	11	0	0
Movie6	2356	3992	0	4	1992	3558	798	0	0	0
Movie7	111	0	0	0	81	0	30	0	0	0
Movie8	82	0	0	0	57	0	25	0	0	0

pair is extracted from the beginning of the pool. A separate thread is forked, which initiates a TCP connection to the peer.

If a TCP connection can be successfully established, the crawler thread then sends a BitTorrent handshake message to the peer indicating that it is an Azureus peer. If the peer is also an Azureus peer (which is determined from the handshake reply received from the peer), the thread speaks to the peer using the Azureus messaging protocol. An interesting feature of Azureus is that Azureus clients have the feature of exchanging gossip messages with each other for exchanging peer lists. Hence, by acting as a an Azureus peer, the crawler thread is able to gather additional IP addresses from the gossip messages that it gets from the Azureus peer. It is also possible to obtain peer lists by accessing a DHT created with Azureus clients, but we do not consider this feature in this study. The new pairs gathered via gossip are again added to the end of the pool.

A separate thread is forked for each IP and port pair in the pool and each thread runs until either there is error in the TCP connection with the peer, or the timer for the peer expires. Similarly, the whole crawling process is continued until the timer for the crawler expires. We tested our crawler on a number of torrents and observed that even for a torrent size as large as 12,000, the crawler was able to crawl more than 90% of total peers within 8 minutes (the total number of peers in the torrent was determined from the Web site hosting the .torrent file). In all of our experiments, we ran the crawler for 8 minutes.

5.1.2 Detection of Connection Attack

For the detection of connection-attack peers, the instrumented client initiates TCP connections to IP addresses from the crawler pool. After having established a TCP connection, the instrumented client speaks the Azureus messaging protocol to the peer if the peer is an Azureus peer, and the "conventional" protocol in case of other peers. For a peers that is an Azureus client, our client marks it as being "connection-attack peer" if it sends more than one Azureus handshake message. Our client also marks a peer as "non-useful" if either a TCP connection cannot be made to it, or if it does not reply with a BitTorrent handshake message when a TCP connection is established.

5.1.3 Detection of Piece Attack

For the detection of piece-attack peers, the instrumented client establishes TCP connections to peers from crawler pool and speaks the "conventional" BitTorrent protocol to all peers. In addition to marking peers as "non-useful," this client marks a peer as being "piece-attack peer" if the peer sends a fake block.

5.2 Active Measurement Results

We collected torrents for the 20 top box office movies during the time of the experiment. We ran an initial crawling on these torrents and checked the peer lists obtained against the blacklist. Out of the 20 movies, we chose the 3 movies (Movies 1 through 3) that appeared to be heavily attacked (based on the large number of blacklisted peers in the peer lists obtained from the crawler). We also selected 3 other movies (Movies 4 through 6) that appeared to be lightly attacked. For each of the 6 movies, we chose the torrent that showed the highest number of blacklisted peers for the movie. We also selected 2 other movies (Movies 7 and 8) that did not show any blacklisted IP addresses in their peer lists.

Before presenting the measurement results, we briefly explain how these movies typically get into BitTorrent in the first place. Just after the theatre release, someone captures the movie with a camcorder (often the visual quality is good, better than VCR quality). The recording is then used to generate pirated DVDs, which are sold throughout the world. One or more of these DVDs then gets seeded in BitTorrent.

Table 5: Measurement Results for Piece-Attack Peers

Movie ID	Total Peers Crawled		IPs from Blacklist		Non-Useful Peers		Useful Peers	Piece-Attack Peers		
	Tracker	Gossip	Tracker	Gossip	Tracker	Gossip		Tracker	Gossip	IPs from Blacklist
Movie1	104	2284	6	68	75	2260	53	4	17	21
Movie2	604	313	1	72	494	255	168	0	8	8
Movie3	59	524	0	29	41	439	103	0	0	0
Movie4	15	86	0	10	10	77	14	0	0	0
Movie5	22	640	0	1	11	640	11	0	0	0
Movie6	374	884	1	1	292	677	289	0	0	0
Movie7	89	0	0	0	67	0	22	0	0	0
Movie8	114	0	0	0	74	0	40	0	0	0

5.2.1 *Test Results for Connection Attack*

Table 4 shows the test results for connection-attack peer attack. We observe that for the 6 attacked movies, 70% (or more) of the peers crawled are not useful, meaning that they are either not reachable by TCP connections, or do not reply with a BitTorrent handshake message after a TCP connection is made. This result is consistent with our passive measurements in Section 3.

We also observe that for Movie 1, half of the useful peers (those who reply with the BitTorrent handshake message) are chatty. For Movie 5, about 85% of the useful peers are chatty. Interestingly, for Movie 5, none of the connection-attack peers that were detected fall into the blacklist that we have. As a verification, we downloaded the same movie using a real BitTorrent client and found that these IP addresses were indeed chatty. This indicates that static blacklisting is not sufficient for preventing such attacks since the attackers can always change IP addresses. Furthermore, for each movie, we observe a large number of blacklisted IP addresses from gossip. However, not all attack IPs come from gossiping - for Movie 5, there are 11 attacker IP addresses from the tracker and none from gossip.

For Movies 7 and 8, which did not appear to be under attack from the initial crawling, no connection-attack peers were detected and the percentage of non-useful peers is still around 70%.

5.2.2 *Test Results for Piece Attack*

Table 5 shows the test results for piece attack for the same 8 torrents. It can be seen that the number of non-useful IP addresses is again 65% (or more) for the 8 torrents. For Movie 1, almost half of the useful peers were piece-attack peers. Since similar results were seen for the connection attack test, it can be concluded that Movie 1 was indeed "heavily attacked" at the time of our experiments. Interestingly, at that time, it was already over 1.5 months after Movie 1 was released and so the movie was below 3 of the other 5 (attacked) movies in the box office rankings at that time.

We compared the list of connection-attack peers and piece-attack peers that were detected for Movies 1 and 2. We found that for each of these, some of the IP addresses detected as connection-attack peers were also detected as piece-attack peers. This reaffirms our claim that a specific attacker behaves differently for different BitTorrent clients.

In summary, our active measurement apparatus and methodology can quickly determine whether the leechers in a torrent are under a piece or connection attack. We have found that several, but not all, top box-office movies are currently under attack. We have also found that published blacklists do not always cover all the attackers in a torrent. We also observed that the majority of the attack IPs enter the system through gossiping; however, some also enter through trackers.

6. BITTORRENT ATTACKS ON SEEDS

Having looked at attacks against leechers in some detail, we now turn our attention to a natural attack against seeds, where the attacker attempts to make multiple connections to a seed and download as much as possible from the seed. We refer to this attack as the *bandwidth attack*. The goal of this attack is, in the early stage of a torrent, to exhaust the initial seed's upload capacity and/or connection slots. The attack is discussed and modeled analytically in Section 3.1.2.

6.1 Experimental Setup

To measure the effectiveness of the bandwidth attack, we created a private torrent using PL nodes as well as a few university and residential nodes. All the attack tests consisted of a single seed sharing a 100 MB file, 30 leechers attempting to download the file, and a number of attack peers. The 30 leechers ran on 30 different PL nodes; whereas the seed was on a residential node. The attack peers worked by directly connecting to the seed, requesting pieces from it, and never forwarding the received blocks to any other peer in the torrent. To observe the change in effectiveness of the

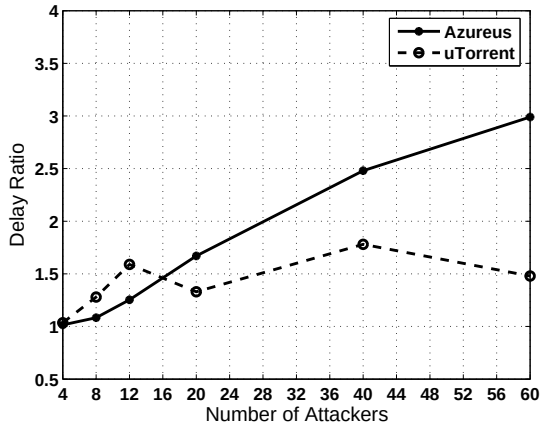
attack, we varied the number of attack peers in each experiment. Furthermore, to simulate an actual torrent, the 30 leechers were all capped with maximum upload and download capacity of 512 kbps and 1 Mbps respectively, and the seed with a maximum upload capacity of 160 kbps. Azureus and uTorrent clients were again tested: in each experiment, the seed used either Azureus 3.0.4.2 or uTorrent 1.7.6. The leechers used the BitTornado 0.3.1.7 client (which was relatively easy to install on PL). In the tests described below, the tests involving 4, 8, and 12 attackers had all the attackers in a university Ethernet network. In order to limit attack traffic coming out of the university network, for tests involving 20, 40, and 60 attackers, the attackers were dispersed in various PL nodes.

All the experiments simulated a flash crowd effect, in that, after the seeding peer started sharing the file, 5 legitimate leechers were started one after another, followed by the attackers, which were in turn followed by the remaining 25 legitimate leechers. In this experiment, we are exploring a best-case scenario for the attackers, in which they quickly detect and join the the new torrent. For the tests involving 40 and 60 attackers, the attackers were launched with a few extra minutes of delay after the first 5 legitimate peers were started. This allowed legitimate peers enough time to find the seed and connect to it, so that not all of the seed's connection slots (50) get grabbed by the attackers, leaving no room for legitimate peers. This mimics the scenario in real torrents: by the time attackers discover the seed, a few leechers will already have connected to the seed and started downloading from it.

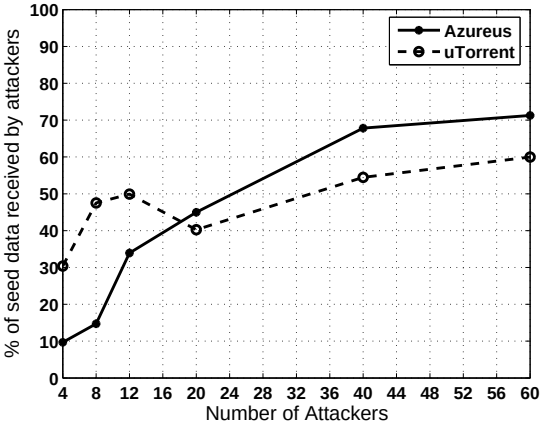
The leechers were configured to record the time taken to download the whole file and the seed was configured to record the total data sent out from it. The effectiveness of each attack experiment was measured in terms of Delay Ratio (ratio of average time to download the file at the leechers with attackers to average download time without attackers) and the percentage of seed data received (wasted) by the attackers. In order to estimate the bandwidth requirement at the attackers, the aggregate bandwidth across all the attackers over each experiment period was used.

6.2 Measurement Results

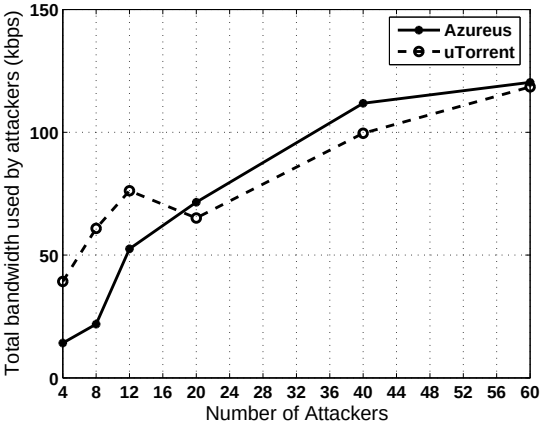
Figure 5 shows the measurement results for the bandwidth attack experiments. Observe from Figure 5(a) that even with the number of attackers as large as 12 (i.e., 40% of the torrent size), the delay ratio for both Azureus and uTorrent seeds never exceeds 1.6. A more considerable value for the delay ratio was observed for Azureus with 40 and 60 attackers. Also, high values for the percentage of seed's data received by attackers and the bandwidth used at attackers reaffirms the fact that they were able to waste considerable amount of seed's



(a) Delay ratio



(b) Percentage of seed data received by attackers



(c) Total bandwidth used at attackers (kbps)

Figure 5: Measurement results for bandwidth attack against Azureus and uTorrent seeds.

resources. However, having more attackers than legitimate peers in a torrent is not feasible, especially when the torrent size is in the order of thousands.

Looking into the source code for Azureus 3.0.4.2 reveals that it does not use a pure bandwidth first algorithm when seeding. The main difference lies in the creation of the list of peers to be unchoked next. Instead of unchoking peers simply based on their download rate from the seed, Azureus creates two sorted lists of interested peers: one is an ascending-ordered list of peers based on their download rate; the other is a descending-ordered list of peers based on the amount of their downloaded data. Azureus sorts the peers again based on the sum of their indexes in the above two lists. The peers at the top of this final sorted list get unchoked. This means that peers with the highest download rates and having fewer bytes downloaded from the seed so far will be given highest priority for unchoking. This explains why the attackers were not able to monopolize the Azureus seed's upload slots with their high download rates. Since uTorrent is closed source, the exact seeding algorithm used is not known. However, looking at the delay ratio values for different experiment sets, we surmise that uTorrent does not use a pure bandwidth first algorithm when seeding files.

In order to verify the feasibility of this attack, we had to find a client that used the conventional bandwidth first seeding algorithm. Looking into the source for the BitTornado 0.3.1.7 client reveals that it does use pure bandwidth first algorithm when seeding. We conducted another test with this seeding client, 30 legitimate leechers, and 12 attackers. As discussed in the mathematical model we constructed for the attack against bandwidth first seeds, in order to make sure that the attack peers get advantage over the legitimate ones when getting unchoked, we set up the upload/download capacities of the seed and the leechers accordingly. The seed was set with a maximum upload capacity of 2 Mbps where as the leechers, dispersed in various PL nodes, were capped with a maximum upload and download capacities of 128 kbps and 256 kbps respectively. The attackers had Ethernet access at our university network ($x \geq \frac{U}{K} > y$, Section 3.1.2). The delay ratio was still very small in this case (1.23). The large values for percentage of seed data received by attackers (92%) and bandwidth used at the attackers (1.84 Mbps) indicates that attackers did get advantage over legitimate peers when getting unchoked. However, since the seed had a large upload capacity, even with only 8% of the seed bandwidth capacity (i.e., around 164 kbps), the attackers were able to quickly download the file (113 minutes).

In summary, even with torrents having relatively large percentage of attackers and only low bandwidth legitimate peers, the bandwidth attack was not able to prolong the download time by a factor of more than 2. Clearly, for torrents containing a combination of both high and low bandwidth legitimate leechers, the download time prolongation will be even less. Therefore, this attack appears to be largely ineffective against BitTorrent seeds (using either the bandwidth first algorithm or its variation), particularly for large torrents. The tests with 40 and 60 attackers are essentially connection-monopolization attacks. These did considerably lengthen the download time for Azureus. However, since most clients today use some combination of the bandwidth first and round-robin seeds, connection monopolization does not seem to be effective in bringing down torrents either.

7. CONCLUSION

We have studied how vulnerable (or resilient) the BitTorrent system is to large-scale, resource-intensive attacks. In doing so, we have identified a number of attacks that can be launched against the four major components of the BitTorrent ecosystem: leechers, seeds, peer discovery, and torrent discovery.

For attacking the leechers, anti-P2P companies are mostly using two different approaches: connection attack and piece attack. Active measurement results on top box-office movie torrents suggest that many (but not all) of these movies are currently being attacked. Passive measurement results show that these two attacks can indeed prolong the average download time of files, particularly for residential broadband users. However, the extent of this prolongation, at least for the torrents studied here, is modest - typically not by more than 50%. Since most BitTorrent users are fairly patient and download files overnight or in the background, we believe that download times need to be tripled to have a significant impact. We have also found that blacklist-based IP filtering is insufficient to filter out all the attackers. To better filter out attackers, it is necessary to design smart online algorithms to identify different types of attackers.

We also carried out measurement studies of attacks against BitTorrent seeds. Even with a reasonably high attacker percentage (e.g., 40%), the download time prolongation for each torrent was not more than 60% for Azureus, uTorrent, and BitTornado seeds. The bandwidth attack therefore does not seem promising for stopping torrents. We also discussed the attack on round-robin seeds, where the main idea is to monopolize the connection slots of the seed with minimal bandwidth. However, since most popular BitTorrent clients today use some combination of bandwidth first and round-robin algorithms, we believe that with minimal bandwidth investment at the seed, the effectiveness of this attack cannot be high enough to largely disrupt a torrent.

We also observed that attacks against leechers and

seeds have to be tailored to the the different client types. This is because the clients employ a wide variety of seeding algorithms; and some clients deviate significantly from the mainline protocols.

The poisoning attack on the tracker should not be very successful either, due to the TCP based messaging between the tracker and peers. The gossip poisoning attack can potentially be effective, but can be easily defended against by removing gossiping. The DDoS bandwidth flooding attack against a torrent discovery site can potentially stop people from finding torrents, but by allocating sufficient bandwidth resources at torrent Web sites, such attacks can be successfully defended against.

Given the resilience of its components against all the major attacks, we believe that the BitTorrent ecosystem is indeed very difficult to stop.

8. REFERENCES

[1] Azureus. `http://azureus.sourceforge.net`.

[2] BitComet. `http://www.bitcomet.com`.

[3] BitTorrent client. `http://www.wikipedia.org/wiki/BitTorrent_client`.

[4] Bittorrent more popular than ever, releases triple in a year. `http://torrentfreak.com/bittorrent-more-popular-than-ever-071009/`.

[5] BitTorrent servers under attack. `http://news.zdnet.com`.

[6] Leaked Mediadefender emails. `http://www.mediadefender-defenders.com/maillist.html`.

[7] Marcovision. `http://www.macrovision.com`.

[8] Media Defenders. `http://www.mediadefender.com`.

[9] P2P monitoring systems. `http://www.evidenzia.de/`.

[10] PeerGuardian. `http://phoenixlabs.org/pg2`.

[11] PPLive. `http://www.pplive.com`.

[12] PPStream. `http://www.ppstream.com`.

[13] Safenet. `http://www.safenet-inc.com`.

[14] SopCast. `http://www.sopcast.org`.

[15] The top 35 torrent sites of 2007. `http://netforbeginners.about.com`.

[16] uTorrent. `http://www.utorrent.com`.

[17] Ziptorrent blacklist. http://torrentfreak.com/ziptorrent-pollutes-and-slows-down-popular-torrents.

[18] Elias Athanasopoulos, Kostas G. Anagnostakis, and Evangelos P. Markatos. Misusing unstructured P2P systems to perform DoS attacks: The network that never forgets. In *Proc. 4th Int. Conference on Applied Cryptography and Network Security*, Singapore, June 2006.

[19] Anirban Banerjee, Michalis Faloutsos, and Laxmi Bhuyan. Is someone tracking P2P users? In *Proc. IFIP Networking*, Atlanta, GA, May 2007.

[20] Nicolas Christin, Andreas S. Weigend, and John Chuang. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proc. ACM EC*, Vancouver, Canada, June 2005.

[21] Bram Cohen. Incentives build robustness in bittorrent. In *Proc. P2PEcon*, Berkeley, CA, June 2003.

[22] Prithula Dhungel, Di Wu, Brad Schonhorst, and Keith W. Ross. A Measurement Study of Attacks on BitTorrent Leechers. In *Proc. IPTPS*, Tampa Bay, FL, February 2008.

[23] Dan Dumitriu, Edward W. Knightly, Aleksandar Kuzmanovic, Ion Stoica, and Willy Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. In *Proc. ACM SIGMETRICS*, Banff, Alberta, Canada, June 2005.

[24] Seung Jun and Mustaque Ahamad. Incentives in Bittorrent induce free riding. In *Proc. P2PEcon*, Philadelphia, PA, August 2005.

[25] Marlom A. Konrath, Marinho P. Barcellos, and Rodrigo B. Mansilha. Attacking a swarm with a band of liars: evaluating the impact of attacks on bittorrent. In *Proc. IEEE P2P*, Galway, Ireland, September 2007.

[26] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, and Lixia Zhang. Clustering and sharing incentives in bittorrent systems. In *Proc. ACM SIGMETRICS*, San Diego, CA, June 2007.

[27] Jian Liang, Rakesh Kumar, Yongjian Xi, and Keith W. Ross. Pollution in P2P file sharing systems. In *Proc. IEEE INFOCOM*, Miami, FL, March 2005.

[28] Jian Liang, Naoum Naoumov, and Keith W. Ross. The index poisoning attack in P2P file-sharing systems. In *Proc. IEEE INFOCOM*, Barcelona, Spain, April 2006.

[29] Naoum Naoumov and Keith W. Ross. Exploiting P2P systems for DDoS attacks. In *Proc. InfoScale*, Hong Kong, May 2006.

[30] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent? In *Proc. NDSI*, Cambridge, MA, April 2007.

[31] Johan A. Pouwelse, Pawel Garbacki, Dick H.J. Epema, and H.J. Sips. The BitTorrent P2P file-sharing system: Measurements and analysis. In *Proc. IPTPS*, Ithaca, NY, February 2005.

[32] Michael Sirivianos, Jong Han Park, Rex Chen, and Xiaowei Yang. Free-riding in bittorrent networks with the large view exploit. In *Proc. IPTPS*, Bellevue, WA, February 2007.

[33] Xin Sun, Ruben Torres, and Sanjay Rao. DDoS attacks by subverting membership management in P2P systems. In *Workshop on Secure Network Protocols (NPSec)*, Beijing, China, October 2007.