1-1-2011

Harnessing the power of BitTorrent for distributed denial-of-service attacks

Lei Wu
University of Central Florida

Jerome Harrington
University of Central Florida

Corey Kuwanoe
University of Central Florida

Cliff C. Zou
University of Central Florida

SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2011; **4**:860–870
Published online 26 July 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.215

RESEARCH ARTICLE

Harnessing the power of BitTorrent for distributed denial-of-service attacks

Lei Wu, Jerome Harrington, Corey Kuwanoe and Cliff C. Zou*

School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, U.S.A.

ABSTRACT

BitTorrent is a popular peer-to-peer (P2P) file-sharing protocol that utilizes a central server, known as a 'tracker', to coordinate connections between peers in a 'swarm', a term used to describe a BitTorrent *ad-hoc* file sharing network. The tracker of a swarm is specified by the original file distributor and trusted unconditionally by peers in the swarm. This central point of control provides an opportunity for a file distributor to deploy a modified tracker to provide peers in a swarm with malicious coordination data, directing peer connection traffic toward an arbitrary target machine on an arbitrary service port. Although such an attack does not generate huge amount of attack traffic, it would set up many connections with the victim server successfully, which could cause serious denial-of-service by exhausting the victim server's connection resource. In this paper, we present and demonstrate such an attack that is entirely tracker-based, requiring no modifications to BitTorrent client software and could be deployed by an attacker right now. The results from both emulation and real-world experiments show the applicability of this attack. Due to the skyrocketing popularity of BitTorrent and numerous large-scale swarms existed in the Internet, BitTorrent swarms provide an intriguing platform for launching distributed denial-of-service (DDoS) attacks based on connection exhaustion.

KEYWORDS

BitTorrent; distributed denial-of-service attack; peer-to-peer networks

***Correspondence**

Cliff C. Zou, School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, U.S.A.
E-mail: czou@cs.ucf.edu

1. INTRODUCTION

Over the past several years, peer-to-peer (P2P) networks have enjoyed a tremendous rise in popularity, primarily as a means of transferring large files over the Internet. In 1999, 'Napster' became the first P2P file-sharing network to attract mainstream attention and was widely used to share music *via* the Internet. The Napster network makes use of a centralized server to maintain a list of currently connected clients and the files that each client makes available at a given point in time [1].

The next P2P file-sharing protocol to garner a great deal of attention was 'Gnutella' [2]. Unlike Napster, Gnutella provides a true P2P network that does not need centralized servers for client tracking. Instead, a Gnutella client requires only the address of a single remote peer to bootstrap its connection to the Gnutella network, obtaining the identities of further peers by querying the peer or peers used during the bootstrap phrase [2].

Following Gnutella, the next wave of P2P file sharing came in the form of the 'FastTrack' protocol, which implemented a supernode-based architecture [3]. A supernode is a high-powered, well connected client in the P2P network that can assume the functionality of a directory server for a number of lower-powered or lesser-connected clients, relieving overheard from those machines and allowing for greater scalability. The most popular of the FastTrack-based networks, Kazaa, achieved a great deal of popularity around 2003 [3].

'BitTorrent' is a different P2P protocol for sharing large files over the Internet created by Bram Cohen [4]. Because of its file-centered design and its fairness mechanism that rewards users for up sharing [5], BitTorrent is very efficient in transferring large files among peers, and hence, has gained its popularity in the last several years [38].

However, BitTorrent has a serious vulnerability that has not been discovered before. BitTorrent utilizes a central server, known as a 'tracker', to coordinate connections between peers in a 'swarm', a term used to describe a BitTorrent *ad-hoc* file sharing network for a file (or a set of files) provided by a file distributor. The tracker of a swarm is specified by the swarm's original file distributor. All peers in the swarm trust the tracker without implementing any authentication or verification procedures. This central point

of control provides an opportunity for a file distributor to deploy a modified tracker to provide peers in a swarm with malicious coordination data, directing peer connection traffic toward an arbitrary target machine on an arbitrary service port.

In this paper, we present a potential attack that exploits the above BitTorrent vulnerability. By deploying such an attack, a malicious attacker could use a popular file (such as a pirated movie) as a bait to launch an application-level, connection exhausting distributed denial-of-service (DDoS) attack using the members of a BitTorrent swarm. This BitTorrent-driven attack does not require any modifications to the client-side BitTorrent software, and hence, it could be immediately implemented in the current BitTorrent world by an attacker. Instead of sending out attack traffic from an attacker's compromised computers, the actual DDoS attack traffic is initiated by the large number of innocent peers within a swarm, which makes the attack efficient, easy to implement, and hard to defend. Furthermore, the BitTorrent attack causes real, complete TCP connections to be made to an arbitrary service port specified by an attacker, allowing the attack to be adapted to a range of target services such as HTTP and SMTP.

The rest of this paper is organized as follows. The related work is introduced in Section 2. We introduce BitTorrent and describe its architecture in Section 3. In Section 4, we present details of the vulnerability inherent in the BitTorrent architecture, and provide the theory behind our DDoS attack. Then, in Section 5, we evaluate our BitTorrent-driven attack, both by emulations and a small-scale real world experiment. We look at some possible defense tactics against the presented attack in Section 6. Finally we summarize this paper in Section 7 and discuss some opportunities for future work.

2. RELATED WORK

Many people have studied how to secure a P2P network so that the network can run normally when it is under attack, either from outside machines or from malicious members in the P2P network. Wallach [6] presented a survey of P2P network security including routing protocol, fairness and trust issues. Castro *et al*. [7] presented attacks to prevent correct message delivery in structured P2P overlays and also defenses to these attacks. Douceur *et al*. [8] studied Byzantine fault isolation in a P2P distributed file system. Maniatis [9] and Giuli [10] presented a set of defenses against various denial-of-service attacks in P2P systems. Sun *et al*. [11] presented how malicious nodes in a P2P system could DDoS external nodes. However, they only focused on KAD (a DHT-based file-sharing system) and ESM (a gossip-based video broadcasting system). Our research is about how attackers could use a P2P network to conduct large-scale DDoS attacks to other targets, not the P2P network itself.

Attackers could also take advantage of the P2P network infrastructure to facilitate their attacks to members of a P2P network, such as using a P2P network to propagate an Internet worm. Yu *et al*. [12] presented a P2P-based worm attack and provided its propagation model. Zhou *et al*. [13] presented a self-defense infrastructure inside a P2P network to contain P2P-based worms. A P2P-based worm tries to infect members of a P2P network, which is different from the BitTorrent-driven DDoS attack presented in this paper.

Naoumov and Ross [14] presented how attackers could publish fake index to conduct index poisoning attack on a P2P system. Index poisoning attack is different from our proposed attack in both the attack principle and target membership: index poisoning attack only affects nodes inside a P2P network, while the BitTorrent-based attack introduced in our paper could attack *any* computer in the Internet.

BitTorrent is a special P2P network protocol [5]. Current research on BitTorrent is mainly on modeling and analyzing its robustness, fairness, and performance [15–18]. Liogkas *et al*. [19] presented three selfish-peer exploits and studied BitTorrent's robustness under these exploits. No research has been done on exploiting BitTorrent to attack an arbitrary target.

The BitTorrent-driven DDoS attack we present in this paper utilizes the communication center (i.e., the tracker) of a BitTorrent swarm to direct connections from members of the network to a target. Conceptually speaking, this attacking idea is similar to the idea deployed by the botnet monitoring system presented in Reference [20]: by hijacking the domain name of the communication center of a botnet (the command & control server), the botnet monitor is able to redirect connections from members of the botnet to itself [20].

'DNS reflection attack' [21] is an amplification flooding attack: an attacker sends spoofed DNS queries to many DNS servers, letting those DNS servers generating a larger volume of DNS response traffic to a spoofed victim. Compared to this DNS reflection attack, the proposed BitTorrent attack exhausts connection resource of a victim instead of bandwidth of a victim.

The paper is extended from our previous conference paper [22]. Through further study, we have identified an error in the previous conference version, and discovered a new challenge faced with the proposed BitTorrent attack. In this paper, we have revised the attack strategy and thus overcome the new challenge. Defrawy *et al*. [23] presented 'BotTorrent', which has the similar idea to exploit BitTorrent to launch DDoS attacks. Our work was conducted independently with Defrawy *et al*. [23] at the same time.

3. BACKGROUND ON BITTORRENT

The BitTorrent protocol was first developed by Bram Cohen and originally released in 2001 [24]. BitTorrent differs from earlier P2P protocols because there is not a single BitTorrent network. Instead, smaller *ad-hoc* networks, known as swarms, are formed for each file (or set of files) that is being transferred. The members of each of these swarms regularly announce their presence to a centralized server, or 'tracker', which maintains a list of all currently connected peers, and

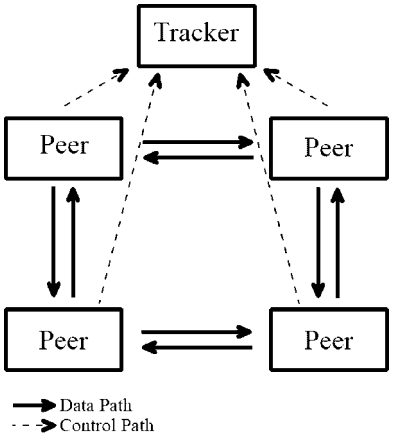

Figure 1. Architecture of a normal BitTorrent swarm.

distributes that list among the peers as they announce to the tracker.

To transfer a file among a group of users *via* BitTorrent, the original distributor of the file must first have a server available that is running BitTorrent tracker software. Then, he or she must generate a 'torrent' file, which contains the URL of the tracker to which peers in the swarm will connect. At this point, the torrent file can now be offered to potential downloaders *via* any normal distribution method. Most often this is done by publishing the torrent file on a popular website.

Figure 1 illustrates the architecture and communication paths of a typical BitTorrent swarm. As seen here, there exists a control path between each peer in the swarm and the centralized tracker server whenever the peer announces itself to the tracker. Any given peer may additionally have data connections with any other peers. The BitTorrent vulnerability is caused by fact that the control channel of a BitTorrent swarm is in the hand of the tracker machine, which can be operated by any user without any verification.

BitTorrent is currently being used for many legitimate and legal purposes, such as distribution of Linux ISO images and software updates for Blizzard's multi-player online role-playing game World of Warcraft. Warner Brothers has made a deal with BitTorrent to distribute and sell over 200 Warner Brothers movies and TV programs through BitTorrent [25]. BitTorrent has arguably received more attention from legitimate interest groups than any previous P2P file sharing protocols or networks. Meanwhile, BitTorrent has become massively popular among the piracy community, being used to transfer music, movies, and television shows *en masse*. Such popular torrent directory websites, such as The Pirate Bay [26] and mininova [27], allow anyone to upload a torrent file, usually anonymously, pointing to an arbitrary tracker specified in the torrent file. Thus, it is possible for a malicious file distributor to create a torrent using a tracker over which he or she has full control, and offer it to a huge pool of potential downloaders in an effective and straightforward manner.

4. VULNERABILITY IN BITTORRENT ARCHITECTURE

4.1. Modifying the tracker

As we have previously seen, peers in a BitTorrent swarm rely on the response provided by the tracker at announce time to determine the identities of the other clients within the swarm and, consequently, what peers they should connect to. Therefore, if an attacker has control of a tracker, he or she can alter the response that the tracker provides to the peers in the swarm.

In our experiments, we made use of an open-source PHP tracker called 'BlogTorrent' [28]. Our modifications were limited to a single function called BTAnnounce() in the tracker program. This function is called each time a client announces to the tracker. The function's pseudo-code is listed below. It demonstrates the primary logic involved in accepting a peer announcement and returning normal output to a BitTorrent client.

```
1. remote_addr = IP address of announcing peer
2. port = BitTorrent port of the announcing peer
3. // split the IP of the announcing peer
4. // into an array of its octets
5. peer_ip_array = split remote_addr on '.'
6. // pack the contents of peer_ip_array into
7. // a binary string of unsigned char
8. peer_ip = pack('C*',peer_ip_array[0],peer_ip_array
   [1],peer_ip_array[2],peer_ip_array[3])
9. // pack the peer's port number into
10. // a binary string of unsigned short
11. peer_port = pack('n*', port)
12. // generate 0–127 based on the current minute
13. time = round_to_int((time() % 7680)/60)
14. // if the peer is a seeder, set the high bit of time
15. if peer is a seeder
16. time = time + 128
17. endif
18. // pack time into a binary string of unsigned char
19. time_out = pack('C', time)
20. // concatenate time, peer_ip, and peer_port to
21. // produce the peer entry in the expected format
22. peer_data = time + peer_ip + peer_port
23. // update or add the peer to the local database
24. if peer is in database
25. update entry to peer_data
26. else
27. add peer to database with peer_data
28. endif
29. // initialize the string to be returned to the client
30. output = ''
31. // concatenate the database entries for all peers
32. // to generate the output to be returned
33. foreach peer
34. output = output + peer
35. endforeach
36. // this is the string in the expected
```

```
37. // format that will be returned to the client
38. return 'd8:intervali1800e5:peers' + length(output)
    + ':' + output + 'e'
```

This function BTAnnounce() serves two main purposes. First, it serves to maintain a local database of currently connected peers within the swarm. Second, it is responsible for creating a peer list to be returned to the calling client, containing the addresses of the currently connected peers stored in the database.

We have modified this function to return our own list of configurable addresses along with the legitimate data returned normally. Given the pseudo-code above, our additions would take place between lines 30 and 31.

Here, we repeat the same process as shown in lines 3–22 to construct a data entry for each of our configured target address and port combinations. Our illegitimate entries are then concatenated and assigned to the output variable before line 31. Having pre-populated the output with connection information for our target or targets, we then proceed to add the legitimate peers to the output string, yielding a binary string to be returned to the BitTorrent client that contains the addresses of the legitimate peers within the swarm as well as the addresses (and service ports) of the arbitrary machine or machines that we have configured to attack.

With the ability to inject arbitrary IP addresses into the peer list, peers in the swarm will now attempt to connect to those targets in an attempt to download chunks of the advertised file. In the next section, we will see how this vulnerability in the BitTorrent architecture may be exploited to stage a DDoS attack.

Figure 2 shows the network architecture of a modified BitTorrent swarm. The modified swarm is able to function normally as a file-transferring swarm since all peers can transfer files to each other normally. In addition, peers within the swarm will make additional connections to our supplied target.

A modified BitTorrent swarm can be configured to attack multiple targets as well. Note that the addresses of these multiple targets may belong to the same server or different machines. If a target has multiple IP addresses assigned to it (e.g., a web farm), this technique can be used to increase the number of connections being made to a target. Alternatively, this could be used to target several distinct machines at the same time.

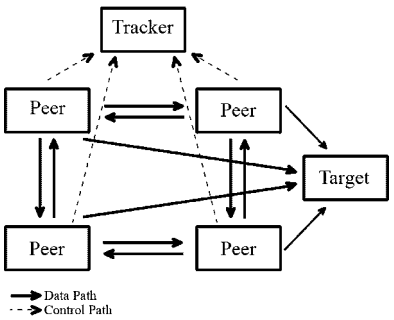

Figure 2. Architecture of our modified swarm (BitTorrent file exchange function is still working, but each peer would generate an additional connection to a victim target on a specified service port).

4.2. A BitTorrent-driven DDoS attack

Given the current usage of BitTorrent and the capability of providing clients with arbitrary addresses in the peer list, it is not difficult to envision a realistic attack scenario. An attacker would first set up a modified tracker, most likely on a server that he or she has previously compromised. Next, the attacker would need to obtain a file or a set of files that are likely to generate high demand—a pirated copy of a blockbuster movie still in theatrical release or a popular new computer game, for example. With these pre-requisites in place, the attacker is now free to generate a torrent file for his or her payload and register the torrent with the modified tracker.

While the attacker could use any means to distribute the newly created torrent, the most straightforward approach would be uploading the torrent to a popular torrent directory, such as Pirate Bay [26]. In this way, the torrent is made available for any user to download freely and join the swarm. At that point, the peers in the swarm that are downloading the file will begin connecting to the supplied target or targets, while still retrieving the torrent payload data normally.

When the number of peers in the swarm with a connection open to an attacked target exceeds the maximum number of connections the target application is configured to accept, the victim service will no longer accept new TCP connections, rendering it unreachable and causing a successful denial-of-service attack.

From the descriptions above, we can see that this BitTorrent-driven DDoS attack has the following properties:

- It requires no modification to BitTorrent client software.
- The attack is hidden from clients since the attack traffic volume from each client is very small and all clients can still upload and download files normally.
- The attack can target multiple victims on arbitrary service ports specified by the attacker.
- The attack does not expose an attacker's real compromised machine (i.e., the tracker) to a victim.
- An attacker can arbitrarily decide the start and end time of a DDoS attack by controlling the tracker's behavior.

5. EVALUATION OF BITTORRENT-DRIVEN DDOS ATTACKS

5.1. Attack emulation

For the emulation, we make use of two different machines. The first machine running Linux Ubuntu 8.04 and Apache

2.2.8 web server acts as our DDoS attack target machine. The second one runs Ubuntu 9.04 with Apache 2.2.11 (to make monitoring easier, we disable the 'deflate' feature in Apache 2.2.11). We modify BlogTorrent tracker and deploy on the second machine, which also supplies the initial seed client and runs a multitude of BitTorrent client 4.0.1 instances (using the command-line client btdownloadheadless.py [29]), simulating the peers within the swarm.

To monitor the number of connections made to the target at any given point in time, a script is written to constantly parse the target machine's Apache server-status page, reporting the number of concurrent requests being served at each parsing time.

A normal execution procedure of a BitTorrent client is as follows:

(1) At the initialization stage, the client contacts the tracker to announce its 'started'.
(2) Download files, and will decide to contact the tracker if certain conditions are met (we will explain the conditions later).
(3) When finishing downloading, it contacts the tracker immediately to announce 'completed'. Now it becomes a seeder, and will contact the tracker periodically to obtain peer list (for the client we tested, the period is 6 min).
(4) If the client process terminates normally, it contacts the tracker to announce 'stopped' just before the termination.

The tracker returns an updated peer list to a client after every announcement (except the 'stopped' announcement) sent by the client. Upon receiving a peer-list from the tracker, a client will contact peers in the peer list. In our tested client program, the maximum number of peers to contact has the default value of 40, and the client will always contact peers in the peer list sequentially. For such client programs, to guarantee that a peer always connects to the DDoS attack target, an attacker just needs to modify the tracker code to put the target IP into the first place of the returned peer list. Some other client programs may have different strategies to select peers in the returned peer list; however, an attacker can always modify the tracker to make sure that clients in his swarm will always contact the target, e.g., by configuring the size of the returned peer list (containing the target IP) to be smaller than the default value of the maximum number of peers to contact used in most BitTorrent client programs.

Through further testing, we find out that the emulation experiment shown in our previous conference paper [22] (Figure 3 in that paper) is not accurate. In the tested BitTorrent client program there is an important parameter named: 'one_connection_per_ip' with a default value of 1, which means 'set up only one connection to each IP address'. This default value works fine for real-world BitTorrent file sharing, but in our emulation the seeder and all clients are running in one machine, and hence, each peer can only connect to one other peer to download files. To make sure all peers can download in the same way as in the real-world BitTorrent network, we set the value of this parameter to 0 to disable this constraint. However, this configuration issue does not affect the real-world experiment shown in the conference paper (Figure 5).

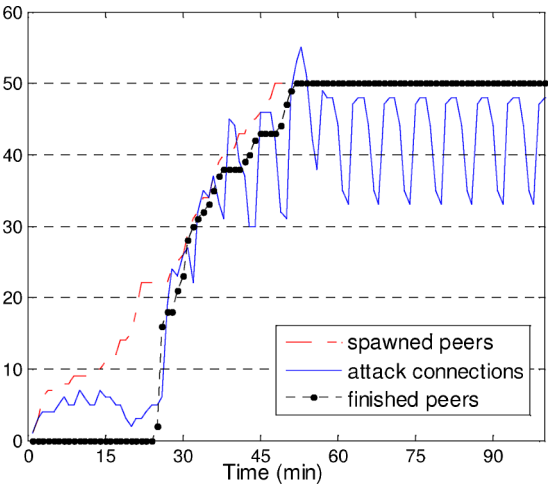

Figure 3. A small-scale emulation with 50 peers (max_upload_rate: 20 KB/s).

We write a client-spawning script to control the generation of client processes in the BitTorrent swarm. Since every client joins in a BitTorrent swarm independently, it is reasonable to model the peers joining process as a Poisson arrival process. In fact, many previous researches [16–18] also modeled BitTorrent system by assuming that the arrival process of downloaders is Poisson process. Thus this script is written to spawn clients following Poisson process.

5.1.1. Small-scale emulation.

First, we conduct small-scale emulation in order to study the attack performance. In the small-scale emulation, we spawn 50 peers following Poisson arrival process with the rate $\lambda = 1/\text{min}$ (i.e., on average one client is spawned in each minute). Additionally, we let another parameter named 'max_upload_rate' to have the default value (20 KB/s), which controls the maximum upload rate for each peer. Figure 3 shows the emulation results. The 'finished peers' shows how many peers have finished download and become seeders at each time.

Figure 3 shows that the number of attack connections does not increase much as more peers are spawned during the first 25 min; it even decreases a little bit. This contradicts with our intuition. We inspect the code of BitTorrent client btdownloadheadless.py, and find out that at the second step of BitTorrent client (shown at the beginning of subsection 5.1), a client contacts the tracker according to the download situation as follows:

1. If (# of connections < min_peers/3) contact tracker every 5 min;

2. If ((# of connections < min_peers) && no downloading happens) contact the tracker every 5 min;
3. If (30 min passed without contacting the tracker) contact the tracker;
4. If (the peer finishes downloading) contact tracker every 5 min.

Where min_peers is a system parameter (default value is 20). Whenever a client contacts the tracker and gets an updated peer list, it will connect to every peer in the peer list, and hence, would set up a connection to the targeted victim until timeout. In Figure 3 the number of attack connections decreases after 15 min because at that moment most spawned peers have content in downloading and their number of active connections bigger than min_peers/3 = 6.7, which means they will not contact the tracker and connect to the target any more for another 30 min.

To check whether the above peer behavior is a common behavior, or a specific one for the particular client code we tested, we analyze the new version of BitTorrent client 5.2.2, and find that the processing procedure is exactly the same using a different default value of min_peers = 40. We also look through some other popular BitTorrent clients, e.g., BitTornado [30], ABC, [31] and Burst [32], and they all have the similar processing procedure.

Figure 3 shows that after 25 min, the number of attack connections goes up quickly. This is because most peers finish downloading and become seeders. Thus they will contact the tracker periodically every 5 min and hence set up connections with the target once in every 5 min.

After 60 min the attack connection curve in Figure 3 exhibits a regular periodical oscillation with the period of 6 min. This is because every peer periodically connects to the target after it becomes a seeder while each peer becomes seeder at different time. The period is 6 min instead of 5 min because the client code checks its status once in a minute.

In Figure 3 most peers quickly finish file downloading and become seeders. In real BitTorrent networks this may not happen often. To simulate the realistic scenario where only a small fraction of peers in a BitTorrent network are seeders and every one has a different network bandwidth, we conduct another emulation by letting the parameter 'max_upload_rate' follow a uniform distribution from 1 to 10 for all clients. Figure 4 shows the experiment results.

Figure 4 shows that before peers finish downloading and become seeder around 60 min, the number of attack connections stays low and even becomes 0. This happens when all peers contact the tracker every 30 min and hence no one is connecting with the target at some time. Of course, that is not what an attacker wants. Therefore, attackers need to revise the tracker's behavior to make sure that most peers contact the tracker every 5 min instead of 30 min.

5.1.2. Attack strategy revision—subgroup-based swarm.

Based on the logic of how a client contacts the tracker (the 4 steps introduce above), attackers can solve the problem

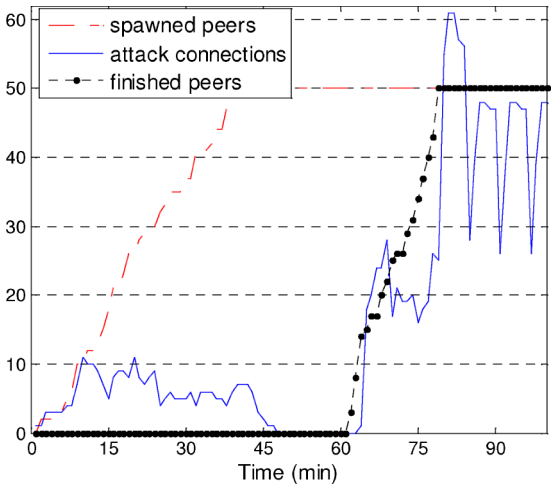

Figure 4. A small-scale emulation with 50 peers (max_upload_rate: uniform distr. 1∼10 KB/s).

illustrated in Figure 4. The problem is caused when most peers satisfy the condition (# of connections > min_peers/3). Since an attacker cannot change client codes, i.e., the value of min_peers cannot be changed, he can change the tracker to make sure that the size of the peer list returned to a peer is smaller than min_peers/3.

To achieve this goal, we modify the tracker code to divide different clients into subgroups, and the returning peer list for each client just contains peers belonging to its subgroup plus the target IP and the original seeder's IP. The group size depends on the value of min_peers in client code (we use 7 because of the default threshold value 20/3 we mentioned before). With this modification of the tracker, Figure 5 shows the results of the experiment. Peers finish downloading with a longer delay because each of them sees a smaller size of swarm. However, every one can normally finish their downloading tasks and the number of attack connections is continuously kept in a high level.

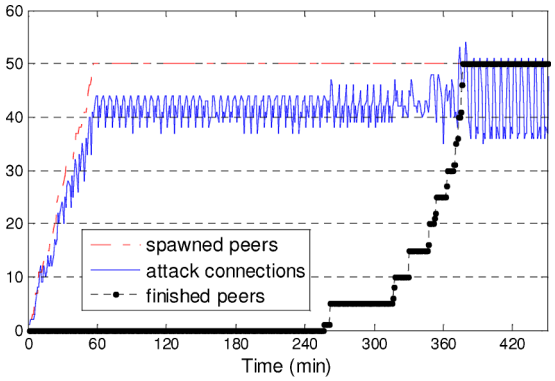

Figure 5. A small-scale emulation with 50 peers (max_upload_rate of the original seeder is 10 KB/s, other peers follow uniform distribution from 1 to 10 KB/s, using the revised subgroup-based tracker code).

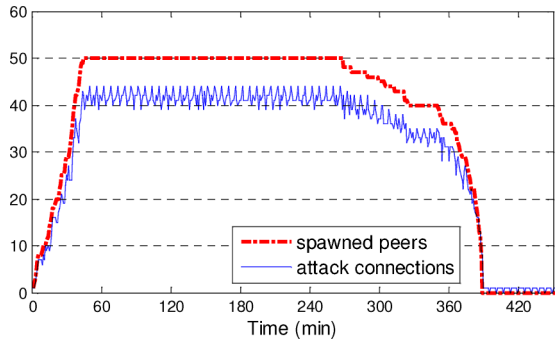

Figure 6. The complete cycle of a small-scale emulation with 50 peers (max_upload_rate of the original seeder is 10 KB/s, other peers follow uniform distribution from 1 to 10 KB/s; use revised subgroup-based tracker code; peers join in following Poisson arrival process and leave following exponential distribution with rate $\lambda = 1$/min after finish downloading).

Now we emulate a complete cycle of attack process by considering that once a peer finishes downloading, it will leave after a while. It is reasonable to assume that once a client finishes downloading, it leaves after an exponential distributed time (with the rate of $\lambda = 1$/min). The result is shown in Figure 6, from which we can see that the curve is well along with our expectation.

With this subgroup-based BitTorrent attack, the attack strength is scalable to large size BitTorrent networks. Each subgroup of a BitTorrent swarm will contribute several attack connections, and subgroups in a swarm attack independently. Therefore, the attack strength is proportional to the size of BitTorrent networks.

5.1.3. Medium-scale emulation.

Now we want to test the attack performance in a medium-scale BitTorrent network with several hundreds of clients. In this experiment, the maximum number of client's processes to be spawned is set to 500. The results are shown in Figure 7.

It would be better if we could conduct a large-scale emulation with a large swarm that has thousands of peers. Unfortunately, our current emulation testbed requires running all client processes on one single machine. Emulation with more than 500 peers would either cause out-of-memory error or crash our testbed machine. A large-scale emulation requires us to generate a brand-new emulation environment and run it on a cluster of machines. We plan to conduct this research as one of our future work.

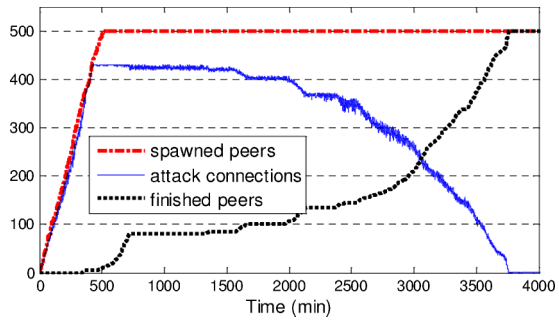

Figure 7. The complete cycle of a medium-scale emulation with 500 peers (all other parameters are unchanged from the experiment shown in Figure 6).

5.2. Peer behavior

Several interesting observations can be made with regard to the behavior of the attacking clients within a swarm. First, an attacking client will hold its connection to a target server open until the server times the connection out. For the standard Apache installation in our experiments, the time-out period is 5 min.

Second, Figures 3–7 show that the average number of attack connections is roughly 80% of the number of peers in the BitTorrent network. This is because each connection to the target web server will last for 5 min before timeout, while a peer contacts the tracker to obtain updated peer list and then connect to the target once in 6 min (1 min used in periodical status checking while 5 min used before contacting the tracker). That means each peer holds one connection to the target in every 5 out of 6 min, i.e., the ratio of the number of active attack connections to the number of peers in the BitTorrent network is $5/6 = 83\%$.

Third, during our emulation, we have also observed that BitTorrent clients re-attempt the connection every time they announce to the tracker. The clients have never blacklisted the target machine, despite the fact that it has never returned a valid response. This is due to the fact that the BitTorrent client software has not considered what to do when receiving an invalid response.

Another issue to be considered is what the actual data exchange looks like between a BitTorrent client and an attacked target. Upon connection to the target server, a BitTorrent client sends a request to the server containing the string 'BitTorrent protocol'. This does provide a possible means of fingerprinting the attack connections.

Figure 8 is a screen capture of a popular BitTorrent client, Azureus [33]. In the screenshot, the client is currently being used as an attacker in our modified swarm. The 'Azureus Peer Details' screen shows this client attacking two separate target addresses, 10.37.129.4 and 192.168.33.16. Simultaneously, the client is connected to a seeding peer, 10.37.129.3 and is transferring the torrent payload successfully. Note that Azureus displays a generic 'Waiting for handshake' error. This error could be caused by a number of things, such as network latency or packet loss. Therefore, even if an end user checks this Azureus Peer Details screen, the user still has no clear clue that any malicious activity is taking place.

5.3. Real world experiment

In addition to our attack emulation and experiments around the behavior of peers in a swarm, we conducted a small-

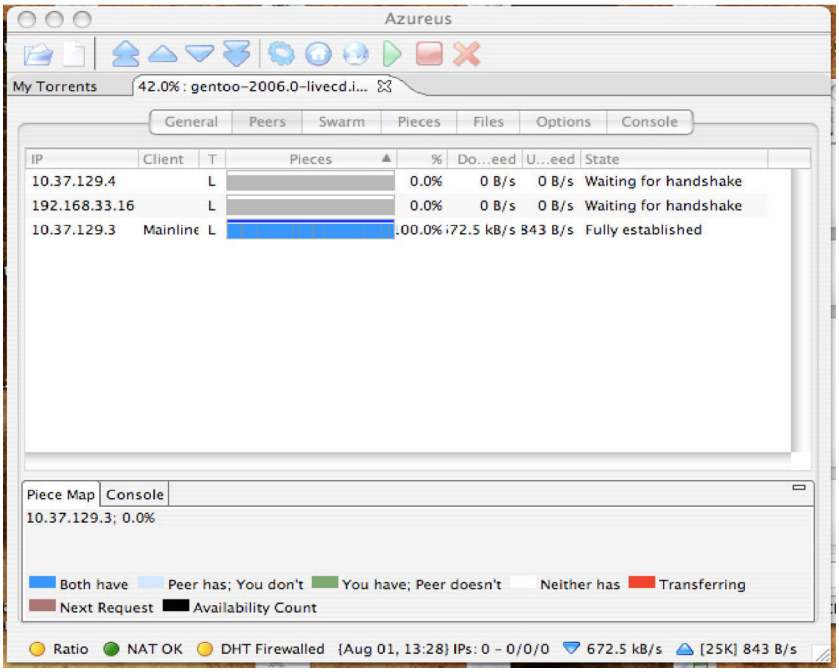

Figure 8. A screen shot of a peer client downloading a file normally while sending attacks to multiple targets at the same time.

scale 'real world' test of our method. For this, we set up a modified tracker, generated a torrent file, and registered it with our tracker. The torrent was then given out to a small group of friends, who were informed as to the nature of the experiment they were to be taking part in.

In this real-world experiment, our tracker was configured to provide the peers with the IP address of a web server under our control as the attack target machine. Furthermore, our tracker was configured with a value of 1 for the minimum number of peers required to begin the attack. We were thus able to see attack traffic being spawned immediately upon the swarm becoming active.

Figure 9 illustrates the number of attack connections generated by our BitTorrent swarm during a 24 h window of our swarm's lifetime. Each data point is the average number of attack connections served by the target web server within 1 h. This average was calculated by tracking the number of open connections at an interval of 1 s. This running count was then totaled and divided by 3600 during later analysis to compute the average number of connections over the course of each 1 h period.

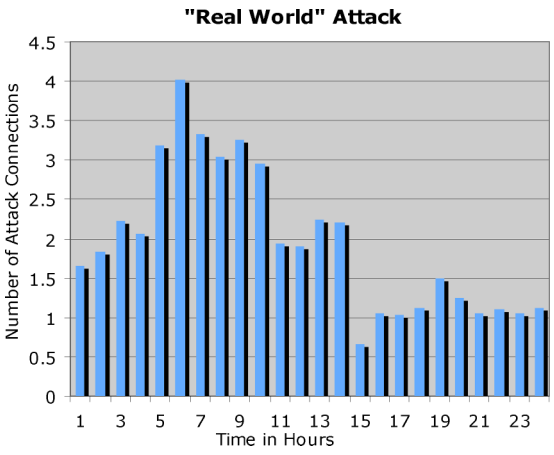

Figure 9. Attack magnitude during a 'real world' experiment.

6. DEFENSE

There are several possible methods that could be used to defend against the proposed BitTorrent-driven DDoS attack. We introduce them in this section.

6.1. Defense initiated by target

The first possible method of defense initiated by target is for a target machine to detect and block attack traffic. As explained in subsection 5.2, an attack TCP connection coming from a BitTorrent peer does not follow the target service specifications and contains the string 'BitTorrent protocol'. This makes it easy for a target machine to detect BitTorrent-driven attack connections and block further attack connections by using a blacklist.

This traffic filtering defense, however, has its limitations. Given the dynamic nature of BitTorrent swarms, the blacklist-based filtering defense requires the blacklist to be updated frequently in order to block attacks from new BitTorrent peers. In addition, the blacklist could be very large if a large BitTorrent swarm is used in the attack.

Considering the challenges faced by blacklist-based filtering, a more convenient yet still effective defense is to simply drop those detected BitTorrent attack connections without using any blacklist. As we discussed in experiment subsection 5.1, an attack client does not reconnect to a target machine if the previous connection is closed by the target—the client only reconnects after another query to the malicious tracker. Therefore, if the target breaks down malicious connections immediately instead of waiting for the long timeout (most web servers have a default timeout of 5 min), the effectiveness of the connection-based BitTorrent DDoS attack will be greatly reduced.

6.2. Torrent validation

Another alternative for defending against the outlined attack is for BitTorrent directory websites to validate submitted torrent files before making them available to the public. Presumably, this validation would be done by: (1) joining the swarm specified in a torrent file; (2) analyzing the peer lists returned by the tracker over a period of time; and (3) verifying that the hosts returned in the peer lists are in fact legitimate peers.

However, this is not a trivial process, and it is prone to error. In addition, an attacker could set a minimum swarm size that must be satisfied before the tracker begins to behave maliciously. Thus, early in the torrent's lifespan, when verification is most likely taking place, the tracker would behave normally and not raise any alarms during the verification process.

6.3. Authorized trackers

The fundamental cause of the BitTorrent-driven attack vulnerability is that the control channel of BitTorrent (trackers) is not managed by authority; any user can designate any machine to be a tracker machine to control a BitTorrent network. Therefore, an effective defense is to make sure that trackers are managed by trusted entities.

To achieve this goal, BitTorrent community could implement a public-key infrastructure similar to the Internet Web community: any tracker used by a BitTorrent network needs to be certified by one certificate authority (CA), where the information and public keys of CAs are hard-coded to all popular BitTorrent clients. In this way, attackers cannot manipulate the control channel of BitTorrent to initiate the DDoS attacks.

6.4. Strict client protocol checking

Another defense would be for the authors of BitTorrent clients to add an increased level of intelligence to their client software. This would still require that each peer connects to the target once. If a peer receives an invalid response (i.e., not following BitTorrent protocol) from a connected target, the client software on the peer can blacklist that target IP address and not attempt any further connections to the target.

However, such a defense could be effective only if a large portion of BitTorrent client software have been upgraded. This could be hard to realize, because currently there are more than 50 BitTorrent client software existing and used by users [34]. In addition, many users never bother to upgrade their BitTorrent client software as long as their client software works fine in file downloading.

6.5. Disabling malicious trackers

The victim of an attack could attempt to locate the malicious tracker and have it disabled or removed from the network through the collaboration with the network or ISP hosting the tracker machine. The tracker is the single point of failure for the survivability of the DDoS attack introduced in this paper.

As with all DDoS attacks, this is easier said than done. First, the victim needs to identify the tracker based on the limited data transmitted by a BitTorrent swarm (the tracker does not attack a victim directly). Second, shutting down a remote tracker would need collaboration from another ISP, which is a time and resource consuming task.

6.6. Behavioral anomaly detection

We have introduced several deterministic techniques above for detecting and filtering the BitTorrent attack. Another defense method is to use behavioral or probabilistic-based anomaly detection. Researchers have proposed many anomaly detection based DDoS defense systems, such as [35,36]. Ranjan *et al*. [35] presented a 'DDoS-resilient scheduler' to prioritize incoming connections based on a suspicion score assigned to each connection. Kim *et al*. [36] presented a dynamic DDoS defense system 'based on a per-packet score which estimates the legitimacy of a packet given the attribute values it carries.'

Anomaly detection based DDoS defense would also be suitable for detection and defense of the proposed BitTorrent based DDoS attack. It is a rich research area to explore, but we will not discuss further since it is out of the focus of this paper.

7. CONCLUSION AND FUTURE WORK

In this paper, we presented a vulnerability inherent in the BitTorrent architecture which can be leveraged to cause innocent peers within a swarm to make connections to one or more configured target machines on arbitrary service ports. Further, we showed how a malicious attacker might utilize this vulnerability to launch a DDoS attack. Although such an attack does not generate huge amount of attack traffic, it

could cause serious denial-of-service by exhausting a victim server's connection resource.

An attack launched using our method has a number of positive properties from the perspective of attackers. First, the attack does not require any modification to the BitTorrent client software used by peers within a swarm—it only requires a modified tracker to be run. Second, the attack provides a high level of configurability, with the ability to configure an arbitrary number of hosts and arbitrary service ports to be attacked. Third, because the attack does not affect the normal transfer of file-sharing data within a swarm, the attack is stealthy to existing BitTorrent clients, reducing the likelihood that a peer would notice that it was being used to generate attack traffic. Fourth, the attack also hides the identity of the real malicious host, the tracker, from the target machine or machines. Attack victims will only receive connections from the peers within a swarm. Finally, an attacker can arbitrarily start or stop the attack and the peers will respond accordingly upon their next announcement to the tracker.

There are several areas in which additional research is useful. The first area in which our work could be built on is to study the behavior of larger swarms carrying out our attack. Due to limited resources, both the emulation and our real-world experiment were executed with relatively small swarms. It would be useful to stage emulations involving hundreds or thousands of peers, as this would more realistically model the environment in which a live attack would be executed. We are currently working on joining the PlanetLab project [37] and use this large-scale distributed network to conduct a realistic denial-of-service attack experiment.

It may also be possible to extend the techniques presented here to other P2P networks. This technique is potentially applicable to any P2P network that has a centralized control server. It may be possible, for example, to create a modified Gnutella supernode capable of reporting to clients that a given host has a specific file when, in fact, it is the target of an attack.

REFERENCES

1. Carlsson B, Gustavsson R. The rise and fall of Napster–an evolutionary approach, *Proceedings of the 6th International Computer Science Conference on Active Media Technology*, 2001.
2. Ripeanu M. Peer-to-peer architecture case study: Gnutella network, *Proceedings First International Conference on Peer-to-Peer Computing*, August 2001.
3. Barkai D. Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net. Intel Press, 2002.
4. Cohen B. Brian's BitTorrent FAQ and guide. www.dessent.net/btfaq
5. Cohen B. Incentives build robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer Systems*, 2003.
6. Wallach D. A survey of peer-to-peer security issues, *International Symposium on Software Security*, 2002.
7. Castro M, Druschel P, Ganesh A, Rowstron A, Wallach DS. Security for structured peer-to-peer overlay networks, *OSDI*, 2002.
8. Douceur JR, Howell J. Byzantine fault isolation in the Farsite distributed file system, *5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.
9. Maniatis P, Giuli TJ, Roussopoulos M, Rosenthal DS, Baker M. Impeding attrition attacks in P2P systems, *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop: Beyond the PC*, 2004.
10. Giuli TJ, Maniatis P, Baker M, Rosenthal DSH, Roussopoulos M. Attrition defenses for a peer-to-peer digital preservation system, *Proceedings of the USENIX Technical Conference*, 2005.
11. Sun X, Torres R, Rao SG. On the feasibility of exploiting P2P systems to launch DDoS attacks, peer-to-peer networking and applications, Springer, New York, 2009.
12. Yu W, Boyer C, Chellappan S, Xuan D. Peer-to-peer system-based active worm attacks: modeling and analysis, *Proceedings of IEEE International Conference on Communications (ICC)*, 2005.
13. Zhou L, Zhang L, McSherry F, Immorlica N, Costa M, Chien S. A first look at peer-to-peer worms: threats and defenses, *4th International Workshop on Peer-To-Peer Systems (IPTPS)*, 2005.
14. Naoumov N, Ross K. Exploiting P2P systems for DDoS attacks, *Proceedings of the 1st International Conference on Scalable Information Systems*, Hong Kong, 2006.
15. Bharambe A, Herley C, Padmanabhan VN. Analyzing and improving BitTorrent performance, *Proceedings of INFOCOM*, 2006.
16. Qiu D, Srikant S. Modeling and performance analysis of BitTorrent-like peer-to-peer networks, *Proceedings of SIGCOMM*, 2004.
17. Guo L, Chen S, Xiao Z, Tan E, Ding X, Zhang X. Measurements, analysis, and modeling of bittorrent-like systems, *Proceedings of ACM/SIGCOMM Internet Measurement Conference (IMC)*, 2005.
18. Pouwelse J, Garbacki P, Epema D, Sips H. The BitTorrent P2P file-sharing system: measurements and analysis. *4th International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2005.
19. Liogkas N, Nelson R, Kohler E, Zhang L. Exploiting BitTorrent for fun (but not profit), *5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.
20. Dagon D, Zou CC, Lee W. Modeling botnet propagation using time zones, *Proceedings of 13th Annual Network and Distributed System Security Symposium (NDSS)*, 2006.
21. Evron G, Vaughn R. DNS amplification attacks, www.securiteam.com/securityreviews/5GP0L00I0W.html

22. Harrington J, Kuwanoe C, Zou. CC. A BitTorrent-driven distributed denial-of-service attack, *3rd International Conference on Security and Privacy in Communication Networks (SecureComm)*, Nice, France, 17–20 September 2007.
23. Defrawy KE, Gjoka M, Markopoulou A. BotTorrent: misusing BitTorrent to launch DDoS attacks, *Proceedings of the 3rd USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet*, Santa Clara, CA, 2007.
24. Thompson C. The BitTorrent Effect. www.wired.com/wired/archive/13.01/bittorrent.html
25. Helm B. BitTorrent goes hollywood, BusinessWeek, May 2006. www.businessweek.com/technology/ content/may2006/tc20060508_693082.htm
26. thepiratebay.org
27. www.mininova.org
28. www.blogtorrent.com
29. The MST3K BitTorrent Guide. mst3k.booyaka.com/ bittorrent_guide.shtml
30. BitTornado. www.bittornado.com
31. ABC (Yet Another Bittorrent Client). pingpong-abc.sourceforge.net
32. Burst! Alternative Win32 Bittorrent Client. krypt. dyndns.org:81/torrent
33. Azureus: Java BitTorrent Client. azureus.sourceforge.net
34. Comparison of BitTorrent software. en.wikipedia.org/wiki/Comparison_of_BitTorrent_software
35. Ranjan S, Swaminathan R, Uysal M, Knightly E. DDoS-resilient scheduling to counter application layer attacks under imperfect detection, *Proceedings of INFOCOM*, Barcelona, Spain, 2006.
36. Kim Y, Lau WC, Chuah MC, Chao HJ. Packetscore: statistics-based overload control against distributed denial-of-service attacks, *Proceedings of INFOCOM*, HongKong, 2004.
37. PlanetLab: an open platform for developing, deploying, and accessing planetary-scale services. https://www.planet-lab.org
38. Naraine R. BitTorrent, Gi-Fi, and other trends in 2004. www.internetnews.com/ent-news/article.php/3294271