Network Management through BitTorrent Blocking and Bandwidth Shaping by ISPs

Gerard Van Wijk
Department of Computer Science
University of Minnesota, Morris
Morris, MN 56267
vanw0067@morris.umn.edu

ABSTRACT

Information sharing is at the heart of the Internet and increasingly prevalent as we go deeper into the 21st century. Of equal importance to Internet Service Providers (ISPs) is the effect of traffic (broadly defined as any information sent over the Internet) on their service. They employ several methods to manage the flow of traffic. BitTorrent is a protocol used to share data over a network. A user will request a file, and peers on the network who have parts of the file will send those parts, while perhaps even receiving the parts they (the peers) are missing. This sharing and peer collaboration allows users to download files quickly without overwhelming the servers where the file originated. As wonderful as this may be, it does not come without consequences. BitTorrent can use up a lot of bandwidth, slowing the network down significantly for any other users.

Although BitTorrent has many legitimate uses (large content updates to computer games, sharing open-source projects, etc.) it also has an unfortunate (but not unfounded) association with illegal file sharing. This has led Internet Service Providers (ISPs) to find ways to mitigate, spread, or even block BitTorrent traffic. These methods are referred to as traffic shaping or blocking. Traffic shaping can also be used to maintain consistent levels of service for customers, whether it is to protect their connection from users who may be using too much or to ensure they receive the amount of data they pay for.

In this paper, we look at the BitTorrent protocol and traffic shaping and blocking protocols (including the token bucket and leaky bucket algorithms). Next we look at the work of Dischinger, et al. who built BTTest, an application used to detect BitTorrent blocking. They found that approximately 8% of BitTorrent interactions are blocked by ISPs. Then we look at the work of Kanuparthy and Dovrolis who built ShaperProbe, an application used to detect token buckets. They found that ISPs are generally honest about how they implement these on their networks.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications; C.2.3 [**Network Operations**]: Network management

General Terms

Measurement, Performance

Keywords

BitTorrent, ISP, bandwidth, shaping, blocking, networks

1. INTRODUCTION

Information sharing is at the heart of the Internet and increasingly prevalent as we go deeper into the 21st century. Of equal importance to Internet Service Providers (ISPs) is the effect of traffic (broadly defined as any information sent over the Internet) on their service. They employ several methods to manage the flow of traffic. BitTorrent is a protocol used to share data over a network. A user will request a file, and peers on the network who have parts of the file will send those parts, while perhaps even receiving the parts they (the peers) are missing. This sharing and peer collaboration allows users to download files quickly without overwhelming the servers where the file originated. This improves upon traditional downloading methods by spreading the burden of proving the desired data from one server to many.As wonderful as this may be, it does not come without consequences. BitTorrent can use up a lot of bandwidth (usually defined as the maximum amount of data that can be sent over an Internet connection), slowing the network down significantly for any other users.

Although BitTorrent has many legitimate uses (large content updates to computer games, sharing open-source projects, etc.) it also has an unfortunate (but not unfounded) association with illegal file sharing. This association comes from the common use of BitTorrent clients in the piracy of media such as music and TV shows. This coupled with the effects of BitTorrent on network performance has led ISPs to find ways to mitigate, spread, or even block BitTorrent traffic [1]. These methods are referred to as traffic shaping or blocking. Traffic shaping is not necessarily associated with BitTorrent and can also be used to maintain consistent levels of service for customers, whether it is to protect their connection from users who may be using too much bandwidth or to ensure they receive the amount of data they pay for [8].

In this paper we will first look at the BitTorrent protocol and how it works in Section 2.1, followed by various methods of traffic shaping and blocking in Section 2.2. Then we will

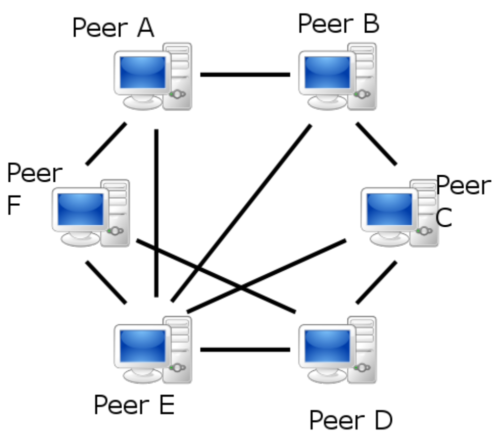

Figure 1: A graphical representation of a peer-to-peer network. Modified from [7].

look at the work of Dischinger, et al. who built an application called BTTest to detect BitTorrent blocking by ISPs in Section3. After that we will examine ShaperProbe, a tool built by Kanuparthy and Dovrolis to detect the presence of a token bucket traffic shaper along a network connection in Section 4. Both of these tools were deployed across the Internet for anyone to use.

2. BACKGROUND

In this section we will look at BitTorrent and traffic shaping and blocking. We will define relevant terms and look at some commonly used algorithms for each.

2.1 BitTorrent

Before distribution, a file transferred using BitTorrent is split into *pieces*, and each piece is split into multiple *blocks*. Although blocks are the transmission unit, peers can only share complete pieces with others. How these are divided is irrelevant to the scope of this paper. The original provider Q of the data then creates the *metainfo file* which provides all the information necessary to download the content and includes the number of pieces, and the IP address and port number of the *tracker*, the only centralized part of the system. A user P will download the metainfo file and contact a tracker. Its purpose is to maintain a list of peers. A *peer* is a user on the BitTorrent network that has all or part of a specific file. The *peer set* is the collection of all peers on a network with a specific file. A peer can be in one of two states: the *leecher* state and the *seed* state. A leecher is still downloading pieces of the content, but is sharing its completed pieces with others. In the *seed* state, it has all the pieces and is sharing them with others. The user P is provided with a set of randomly selected peers, some leechers and some seeders, by the tracker. A representation of a peer-to-peer network can be seen in Figure 1. Peer P will now request different pieces of the data from this set of peers. [4]

Most clients will use a *rarest-first algorithm* to determine which pieces to download. Upon establishing a connection, a *bitfield* message is exchanged between peers containing a list of all the pieces a peer has. The new peer will then use these lists to determine the rarest piece in the peer set and request it first. Once the piece is downloaded, it sends a *have* message to each peer in the peer set. Peers independently maintain a list of the pieces each of their remote peers has and build a *rarest-pieces set* containing the indices of the pieces with the least number of copies. This set is updated every time a remote peer announces that it acquired a new piece, and is used by the local peer to select the next piece to download. This ensures that the rarest piece is always the one that is being requested and reduces the risk of missing pieces should a peer leave the network. [4]

In Figure 1, suppose Peer E just joined the network and is requesting pieces from each other peer. Peer A has the entire file and Peer B is only missing pieces that Peers A and C have. Each of these interactions (lines in Figure 1) can be imagined as being Figure 2.

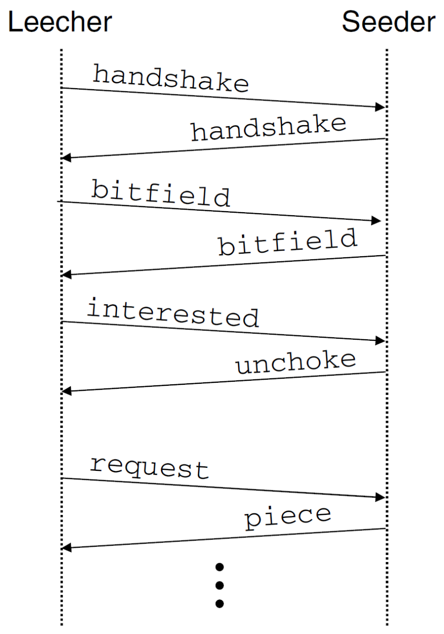

Figure 2: BitTorrent packet exchange. Taken from [1].

To decide which peers to exchange data with, a peer uses the choking algorithm. Peer A is *interested* in peer B when B has pieces of the content that A does not have. Conversely, peer A is *not interested* in peer B when B only has a subset of the pieces of A. Peer A is *choked* by peer B when B decides not to send any data to A. Conversely, peer A is *unchoked* by peer B when B is willing to send data to A. It should be noted that this does not mean that peer B is uploading data to A, but rather that B will upload to A if A issues a data request. This algorithm gives "preference to those peers who upload data [seed] at high rates [4]." Once per *rechoke period* (usually ten seconds) the peer recalculates how much data it receives from each of its peers. It only sends data to a fixed number of the fastest ones for the next rechoke period. Because not all users are required to seed, this algorithm incentivizes seeding. A general representation of a *BitTorrent flow* (a complete exchange of the BitTorrent protocol) between two peers can be seen in Figure 2.

2.2 Traffic Shaping and Blocking

Traffic shaping is used by Internet service providers (ISPs) to manage the flow of *packets* (the units in which information is sent out over the Internet), maintain their bandwidth, and optimize performance [8]. There are several different ways this can be done, including the leaky bucket or token bucket algorithms. Simpler shaping schemes shape traffic by some rate of bits per second while more advanced ones will classify the traffic and shape based on things such as port number or protocol. Put simply, a port is a number that a packet is assigned based on its protocol or the type of information it contains. [1]

ISPs will sometimes inspect the contents of packets using Deep Packet Inspection (DPI). In DPI, a packet is inspected for certain patterns. Some examples of possible patterns are the IP address of the origin and destination of the packet, port number, or a particular line of code. For example, when shaping BitTorrent traffic ISPs might look for information coming from port 6881. This is a well known BitTorrent port, and most clients will use this port. In the detection of any particular protocol or type of packet, there may be thousands of patterns to check. Once the traffic has been identified as something the ISP wishes to block, an RST packet is sent, which prompts the connection to reset, dropping (discarding) any packets still in transit. [1, 5]

2.2.1 *Token Bucket*

Many ISPs offer multiple tiers of service with bandwidth and connection speed varying between tiers. To manage this, some ISPs make use of the Token Bucket algorithm. The analogy of a bucket here can be confusing (especially with the existence of the Leaky Bucket algorithm described in Section 2.2.2) and a more useful analogy may be that of a toll booth. Imagine the cables bringing Internet to your block as a highway for packets to travel on, each lane going to a different house. The middlebox (which could be the part of the network splitting the data within a city block or the ISP's modem installed in a house) acts as a toll booth, charging packets differently depending on which exit they are taking. In order to pass through the toll, a packet must pay a certain number of tokens, which is determined by the packet's size. The toll booths supply the tokens to the packets at a certain number of tokens per second. The rate at which tokens are supplied is determined by the customer's service plan. This provides ISPs with an infrastructure to change the level of service to a particular location without having to rewire any part of their system.

Suppose we have an Internet connection with a maximum speed of C bits per second (bps), and this connection has a token bucket associated with it. The token bucket has a maximum capacity of σ tokens and generates ρ tokens per second, with $\rho < C$. If we assume that one token is worth one bit, a packet of size L bits is only allowed to continue along the connection if the token bucket has at least L tokens. If there are not enough tokens, the packet will wait until there are. Once the packet is sent, L tokens are consumed from the bucket [3].

Suppose we start with a full token bucket (σ tokens), and we get a large burst of packets of size L bits each. k of those packets can be sent immediately at the rate of the maximum speed in C bps, with $k = \frac{\sigma/L}{1-\rho/C}$. In this formula, the numerator simply divides the existing tokens to some of the packets; the denominator accounts for additional tokens

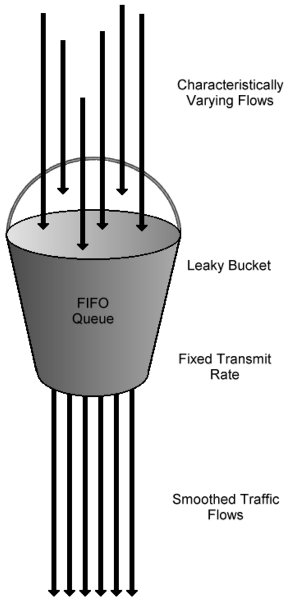

Figure 3: The leaky bucket algorithm as represented on [6].

generated in the time it takes for these first packets to pass through. After those k packets, the remaining packets will be sent at the token generation rate ρ.

For the rest of this paper ρ will be referred to as the "shaping rate", the maximum speed of the connection C will be referred to as the "peak rate", and σ will be referred to as the "maximum burst size". Another way to describe a traffic shaper using the token bucket algorithm is by specifying that the maximum number of bits A that can be sent in any interval of duration τ, starting with a full token bucket is:

$$\hat{A}(\tau) = min\{L + C\tau, \sigma + \rho\tau\}$$

The left side of the *min* function in this formula calculates the maximum number of bits that the network can send in τ seconds; the right side calculates the maximum bits that can pass through the token bucket in τ seconds. In most circumstances, $L + C\tau$ should be *less than* $\sigma + \rho\tau$. In the event that it is not, the token bucket is essentially useless because the hardware on the network is a greater limiting factor than the bucket.

2.2.2 *Leaky Bucket*

The leaky bucket algorithm operates on a First-In First-Out queue. Packets arrive at varying rates and are placed in the bucket (queue). Packets leak out of the bucket (are removed from the queue and sent) at a fixed rate. If the bucket is full or does not have room for an arriving packet, it overflows (is discarded) [6]. Because data can be discarded in a leaky bucket, most ISPs prefer to use token buckets [3].

Token buckets and leaky buckets are mostly used as blanket traffic shaping, with no consideration taken into what kind of data is being passed.

2.2.3 *Net-Neutrality Concerns*

There has been considerable debate regarding the legitimacy of traffic shaping and blocking. There have been concerns that several of the methods described in this paper are in violation of net-neutrality principles. According to these principles, it is important that the flow of information over the Internet be free, unhindered, and "not discriminate[d] on the basis of source, destination, or ownership" [2].

However, most ISPs would argue that they are simply reasonably managing their networks. The Federal Communications Commission (FCC) decided that net-neutrality is important, but it is also important for ISPs to maintain their quality of service. Their classification of the issue has remained vague, so Jordan and Ghosh [2] tried to tackle the definition of reasonable/unreasonable network management.

DPI employed in blocking BitTorrent traffic was deemed unreasonable. This practice terminates the connection and is applied regardless of the end-user's wishes, both of these factors raising red flags. Traffic shaping using a token bucket or similar method was seen as reasonable. Although the hardware is being forced to under-perform, the level of performance being achieved was agreed upon by the end-user; this level of transparency is what allows this practice to be net-neutral. Although Jordan and Ghosh did not examine leaky buckets, the fact that data can be lost would deem this an unreasonable management practice. [2]

3. BTTEST

Dischinger, et al. built a Java based web-application called BTTest for the purpose of studying BitTorrent traffic blocking practices by various ISPs. In order to maximize diversity in the set of results, BTTest was promoted by word of mouth on the Internet. Users could launch BTTest and it would simulate communication with a BitTorrent server. They ended up with 47,300 end-users using 1,987 different ISPs.

3.1 Methodology

Once a user launched the application, they communicated with a BTTest server (hosted on a network that was known not to block BitTorrent) and a series of BitTorrent flows was emulated. BTTest closely monitored both ends of communication, and if the flow was interrupted it checked the packets for RST (connection reset) packets. These were considered by Dischinger, et al. to be evidence of blocking.

BTTest used the standard BitTorrent protocol described in Section 2.1 and would run multiple flows with different parameters in order to determine how ISPs detect BitTorrent traffic. Dischinger, et al. varied three values. The first is TCP Port: half of the flows used port 6881, which is commonly known as a BitTorrent port. The other half used 4711, which has no associations. The second is the direction of the flows, with half going from the server to the user, and the other half from the user to the server. The third is protocol: half of the flows contained BitTorrent packets, while the other half were filled with random bytes. On each test they ran each possible combination of these parameters twice, for a total of sixteen flows.

By varying these parameters, they were able to determine not only how ISPs identify BitTorrent traffic, but also if the network connection was stable enough to be considered in the final results. Data sent from port 4711 with random bytes was used as a sanity check. If it was dropped, there was assumed to be a problem with the connection and the test was removed from the results.

BTTest would take special notice of any flows in which an RST packet was present before all of the data was sent back and forth. This indicates that the ISP identified BitTorrent traffic and forged an RST packet to forcefully close communication.

ISP	# measured hosts	# blocked hosts
Comcast	4397	2574
Cox	1004	508
RoadRunner	2086	50
MediaCom	120	17

Table 1: A brief summary of the results of Dischinger, et al. More can be found at [1].

3.2 Results

Dischinger, et al. collected 47,318 result sets from 1,987 ISPs world-wide. Of these, 146 did not contain results for all 16 flows. Another 17 failed during at least one of the sanity checks, and these were removed. Some users ran BTTest multiple times. To avoid bias in the results, they only considered the first successful result set from each IP address. After removing either duplicate or failed results, they had a pool of 41,109 result sets.

Dischinger, et al. found evidence of BitTorrent blocking in 8.2% (3,353) of their results; most of these were located in the United States and Singapore. 44.3% of their users were located in North America, 26.7% in Europe,and 17.9% in South America. 47 (0.02%) of the tested ISPs showed evidence of blocking. These numbers indicate that a majority of the blocking came from a small number of ISPs and that no ISP blocked BitTorrent universally. Comcast and Cox show the greatest amount of blocked BitTorrent interactions, as seen in Table 1.

They provide a few explanations for the lack of universal BitTorrent blocking. Middleboxes that perform the blocking may not be deployed over each of an ISPs network paths. Alternatively, they may only begin to block BitTorrents as soon as the network load reaches a certain point. Some ISPs may even allow a certain amount of BitTorrent traffic and block only those who pass a certain threshold.

Port number proved to be negligible in traffic identification, with only 15.8% (530) of the blocked result sets showing evidence of this regardless of the content or direction of the flows. On the other hand, 99.5% (3,335) of the blocked sets showed evidence of blocking upstream (traffic going from a computer out to the Internet) with only 2.1% (71) blocking in the downstream (traffic going from the Internet to a specific computer) direction. Finally, 98.2% (3,293) of the blocked result sets showed blocking based on BitTorrent messages.

From this information, Dischinger, et al. infer that ISPs use deep packet inspection to identify users uploading BitTorrent files (regardless of the port they are using) and block these flows.

4. SHAPERPROBE

Kanuparthy and Dovrolis [3] built another tool with the goal of detecting traffic shaping called ShaperProbe. The primary difference between ShaperProbe and BTTest is that ShaperProbe detects token buckets instead of forged RST packets.

4.1 Methodology

ShaperProbe is a downloadable application that will detect a token bucket on a network and then infer its characteristics (the speed in bps, the size of the bucket in tokens, and the token generation rate). The application uses an end-to-end communication network where one end (SND) sends packets to the other (RCV). The presence of traffic shaping is determined by RCV.

ShaperProbe estimates the narrowest bandwidth in the connection in bps and probes at a *constant bit rate* $R_s = C$. Upon reception of packets, RCV timestamps them and constructs a timeseries (a graph which is not too different from Figure 4) $R_r(t)$ of the rate at which packets are received. The timeseries is separated into intervals of size Δ (represented by points in Figure 4). The ith interval will contain all packets received in the interval $[(i-1)\Delta, i\Delta]$. The timeseries $R_r(i)$ is estimated as the total bytes received in interval i divided by Δ [3].[1] If there is a token bucket traffic shaper on SND $\rightarrow$ RCV, there is a value at which $R_r(i)$ shows a significant change (referred to as a level shift and represented by τ in Figure 4) in the rate data is received.

Their method of detection relies on numbering each interval according to the rate at which data was received during that interval. For example, the slowest interval is assigned the number 1 and the next slowest is assigned 2. The intervals remain organized chronologically. This method softens the blow of outliers. These numbers (ranks) are calculated online in real-time which means that at the start of each new interval, the ranks of the previous intervals are updated.

They use τ to identify the start of a level shift if it is the first index to satisfy three conditions and n is the number of intervals whose received rate value have been calculated so far.

1: All ranks to the left of τ are *greater than or equal to* all ranks to the right of τ.

$$\min_{i=1\ldots\tau-1} r(i) \geq \max_{j=\tau+1\ldots n} r(j)$$

The left side of this function calculates the lowest received rate to the left of τ while the right side calculates the highest received rate to the right of τ.

2: There is a minimum time duration before and after the current rate measurement:

$$n_L < \tau < n - n_R$$

n_L is determined based on typical durations of data bursts by ISPs and n_R is a sanity check to make sure the drop is not just a temporary variation. This helps to ensure that the level shift is actually attributable to a token bucket.

3: There must be a significant drop in the median received rate at point τ:

$$\tilde{R}_r(i)_{i=1\ldots\tau} > \gamma \tilde{R}_r(j)_{j=\tau\ldots n}$$

$\tilde{R}_r$ is the median of all received rates calculated so far and γ is a suitable threshold based on "empirical observations of ISP capacities and shaping rates in practice" (see Section 2.3 of [3]).

Another index β is used to indicate the *end of a level shift*: $\beta \geq \tau$ and β is the last point to satisfy the first condition.

[1]Kanuparthy and Dovrolis note a potential error in the estimation of $R_r(t)$ up to $\epsilon = \pm S/\Delta$ where S is the size of the largest packet the network can send. They minimize this by choosing a reasonably large Δ.

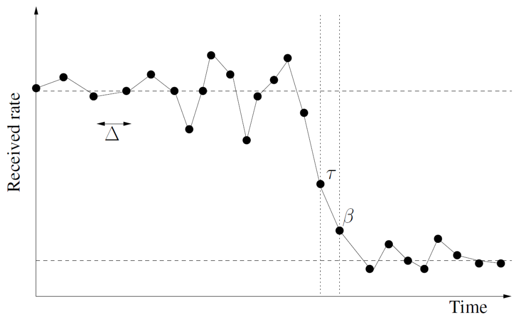

Figure 4: Active probing: Level shift detection [3].

Once a level shift has been detected, the properties of the token bucket are estimated. The estimated token generation rate $\hat{\rho}$ is the *median* of the received rates after β:

$$\hat{\rho} = \tilde{R}_r(i)$$

This is because once the bucket has run out of tokens, information can only pass through once there are enough tokens to let it through. At this point, information will continue to pass at rate ρ until there is a pause that allows the token bucket to fill.

The estimated size of the bucket $\hat{\sigma}$ is a range based on the number bytes sent until the τth interval, their estimate of $\hat{\rho}$, and the received rates:

$$\hat{\sigma} = \sum_{i=1}^{\tau} [R(i) - \hat{\rho}]\Delta \pm \frac{[R(i) - \hat{\rho}]\Delta}{2}$$

This will calculate the difference between the rate at which data was received and the estimated token generation rate for each interval before the level shift at τ. This accounts for tokens that are being generated at that time and only counts the extra tokens which were not generated. Because this is only an estimation, their range ends up being 1/2 of the extra tokens for the lower bound and 3/2 of the extra tokens for the lower bound.

4.2 Results

Kanuparthy and Dovrolis tested the accuracy of ShaperProbe on two ISPs for which they knew the shaping properties and connection speeds. One of these was a network on Comcast with 10Mbps up (from the end-user to the Internet) and 22Mbps down (from the Internet to the end-user) shaped to 2Mbps up and 12Mbps down. Over 60 runs on this network, ShaperProbe failed to detect traffic shaping twice. A second network (a RoadRunner network) showed evidence of shaping downstream with very little (if any) shaping upstream. Another network (an AT&T network) did not use any traffic shaping. Over 60 runs on this network, there were no detection errors.

They measured the results from ShaperProbe runs against ISPs' advertised shaping rates.

4.2.1 Comcast

Comcast uses the PowerBoost service to temporarily increase bandwidth for users who are downloading or uploading large files. Across their different tiers of service ShaperProbe detected that the most common burst sizes upstream were 5 Megabytes per second (MBps) and 10 MBps. They

C (MBps)	ρ (MBps)	σ (MB)	Burst duration (s)
.4375	.125	5	16.7
.6	.25	5, 10	15.2, 30.5
1.1	.6875	10	25.8
1.8125	1.25	10	18.8

(a) Upstream

C (MBps)	ρ (MBps)	σ (MB)	Burst duration (s)
2.425	.8	10	6.4
2.6375	1.6	10	10.1
3.525	2.125	20	14.9
4.3	2.925	20	15.3

(b) Downstream

Table 2: Detected shaping rates from Comcast. In these tables C is the maximum bandwidth, ρ is the shaping rate, and σ is the maximum burst size. The rows represent different tiers of service. Modified from [3].

found the downstream rates to be 10 MBps and 20 MBps. Under this system, users who typically experience a download speed of 1.6 MBps (ρ [MBps] in Table 2.b) can get a boost for 10.1 seconds (Burst duration [s]) to 10 MBps (σ [MB]). Their findings match Comcast's advertising. It should be noted that this boost is still limited by the maximum bandwidth of the network.

4.2.2 RoadRunner

RoadRunner is an interesting case. According to Kanuparthy and Dovrolis, RoadRunner only advertised shaping downstream but found no advertisement of shaping upstream. On RoadRunner networks, ShaperProbe detected shaping in only 6.5% of the upstream runs and 63.9% of the downstream runs, confirming RoadRunner's claims. Kanuparthy and Dovrolis operate under the hypothesis that RoadRunner does not shape its upstream traffic and that they have a 6.5% false positive detection rate. This is unusual because most ISPs tend to implement the same amount of shaping in either direction.

4.2.3 AT&T

Kanuparthy and Dovrolis found no mention on AT&T's website of traffic shaping. However, 10% of ShaperProbe runs on AT&T networks detected traffic shaping. In their paper they say that these were "*probably mis-diagnosed* as shaping." Upon further investigation, however, they found that about a third of the positive detections had very similar shaping rates. Of these, they found that 80% came through MediaCom networks. This could indicate that these are not false-positives after all and although AT&T does not practice traffic shaping, other ISPs that it serves (such as Mediacom) could practice traffic shaping. These ISPs receive Internet service from AT&T and redistribute this service.

5. CONCLUSIONS

In this paper, we examined network management protocols employed by ISPs, particularly BitTorrent blocking and traffic shaping protocols. We looked at the work of Dischinger, et al. who built BTTest, an application used

ISP	Upstream%	Downstream%
Comcast	71.5 (34874)	73.5 (28272)
RoadRunner	6.5 (7923)	63.9 (5870)
AT&T	10.1 (8808)	10.9 (7748)

Table 3: These results from [3] show percentage of runs that detected traffic shaping.

to detect BitTorrent blocking. They found that approximately 8% of BitTorrent interactions are blocked by ISPs. It is interesting to note that Comcast implemented BitTorrent blocking in 2007 [2], Dischinger, et al. conducted their study in 2008, and it was concluded by Jordan and Ghosh in 2010 that this kind of traffic blocking is unreasonable. I ran BTTest on a Comcast network on November 25, 2012 and found no evidence of BitTorrent blocking.

Next we looked at the work of Kanuparthy and Dovrolis, who built ShaperProbe, an application used to detect token buckets. They found that ISPs are generally honest about how they implement these on their networks. This is likely because token buckets are regarded as reasonable network management and their effects are agreed upon by the end-user.

6. REFERENCES

[1] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi. Detecting bittorrent blocking. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 3–8, New York, NY, USA, 2008. ACM.

[2] S. Jordan and A. Ghosh. A framework for classification of traffic management practices as reasonable or unreasonable. *ACM Trans. Internet Technol.*, 10(3):12:1–12:23, Oct. 2010.

[3] P. Kanuparthy and C. Dovrolis. Shaperprobe: end-to-end detection of ISP traffic shaping using active methods. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement*, IMC '11, pages 473–482, New York, NY, USA, 2011. ACM.

[4] N. L. E. K. Legout, Arnaud and L. Zhang. Clustering and sharing incentives in BitTorrent systems. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):301–312, 2007.

[5] P. Piyachon and Y. Luo. Efficient memory utilization on network processors for deep packet inspection. In *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, ANCS '06, pages 71–80, New York, NY, USA, 2006. ACM.

[6] Wikipedia. Leaky bucket — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Leaky_bucket&oldid=517425558`, 2012. Online; accessed 10-October-2012.

[7] Wikipedia. Peer-to-peer file sharing — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Peer-to-peer_file_sharing&oldid=507261462`, 2012. Online; accessed 10-October-2012.

[8] Wikipedia. Traffic shaping — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Traffic_shaping&oldid=514675409`, 2012. Online; accessed 10-October-2012.