Detecting BitTorrent Blocking

Marcel Dischinger[†] Alan Mislove[†‡] Andreas Haeberlen[†‡] Krishna P. Gummadi[†]

[†]MPI-SWS [‡]Rice University

ABSTRACT

Recently, it has been reported that certain access ISPs are surreptitiously blocking their customers from uploading data using the popular BitTorrent file-sharing protocol. The reports have sparked an intense and wide-ranging policy debate on network neutrality and ISP traffic management practices. However, to date, end users lack access to measurement tools that can detect whether their access ISPs are blocking their BitTorrent traffic. And since ISPs do not voluntarily disclose their traffic management policies, no one knows how widely BitTorrent traffic blocking is deployed in the current Internet. In this paper, we address this problem by designing an easy-to-use tool to detect BitTorrent blocking and by presenting results from a widely used public deployment of the tool.

Categories and Subject Descriptors: C.2.3 [Computer-Communication Networks]: Network Operations; C.2.5 [Computer-Communication Networks]: Local and Wide-Area Networks; C.4 [Performance of Systems]

General Terms: Measurement, Performance, Experimentation

Keywords: BitTorrent, blocking, network measurement

1. INTRODUCTION

Access ISPs like residential cable and DSL providers are increasingly deploying middleboxes, such as traffic shapers, blockers, and firewalls, to monitor and manage their customers' traffic. These middleboxes classify and manipulate flows belonging to different applications according to ISP-specified policies [1, 2]. As traffic management policies are often driven by business interests (e.g., peering or transit agreements), many ISPs do not publicly disclose the details of their middlebox deployments. Thus, end users today may not know about the presence of the middleboxes, and often do not understand the impact of ISP traffic management policies on the performance of their applications.

Recently, it has been reported that certain access ISPs [3, 4] are surreptitiously blocking their customers from uploading data using the popular BitTorrent file-sharing protocol. The ISPs were found to tear down TCP connections identified as BitTorrent flows by sending forged TCP reset (RST) packets to the end hosts. These reports of blocking sparked an intense and wide-ranging policy debate between ISPs, consumer advocacy groups, web site operators, and government agencies on acceptable ISP traffic management practices and network neutrality [5]. However, to date, end users lack access to measurement tools that can detect whether their access ISPs are blocking BitTorrent traffic. As a result, no one knows how widely BitTorrent is blocked in the current Internet.

In this paper, we present a large-scale measurement study of BitTorrent traffic blocking by ISPs. To conduct the study, we designed a tool called BTTest, which enables end users to test for blocking on their own access links. BTTest runs as a Java applet within the user's web browser; it emulates a BitTorrent flow to a server under our control, and it checks whether this connection is aborted with TCP reset packets that neither endpoint has sent. BTTest is *easy to use*, which enables us to gather data about a large number of ISP links. The test achieves *reproducible results* because it runs in a controlled environment, and its analysis is *conservative* in the sense that it checks for a very specific blocking technique, namely interrupting flows with forged connection reset packets.

We deployed BTTest on publicly accessible test servers and invited end users around the world to test their links. Over a period of 17 weeks, more than 47,300 end users in 1,987 ISPs world-wide ran BTTest. We examined the traces gathered during these tests for evidence of BitTorrent blocking. Our findings show that BitTorrent uploads are being blocked for a significant number of hosts, mostly from ISPs located in the USA and in Singapore. While our current study is limited to detecting BitTorrent blocking, it represents a first step towards the broader goal of making ISP policies more transparent to end users.

The rest of the paper is organized as follows. Section 2 provides an overview of the efforts by ISPs to shape BitTorrent traffic and discusses existing work related to detecting such behavior. Section 3 describes the design of our BTTest tool and the methodology used to gather traces at scale. In Section 4, we explain how BTTest analyzes the traces to detect BitTorrent blocking, and Section 5 presents the findings of our measurement study. We conclude in Section 6 with a discussion of open challenges and potential future work.

2. BACKGROUND AND RELATED WORK

BitTorrent [6] is a popular peer-to-peer file-sharing protocol, that accounts for a large and rapidly growing fraction of the data bytes sent over the Internet [7]. The resulting increase in Internet traffic is raising the cost of transit for ISPs, many of which are selling flat-rate plans with unlimited Internet access to their customers. Thus, it is not surprising that an ISP would implement strategies to reduce the amount of BitTorrent traffic generated by its customers.

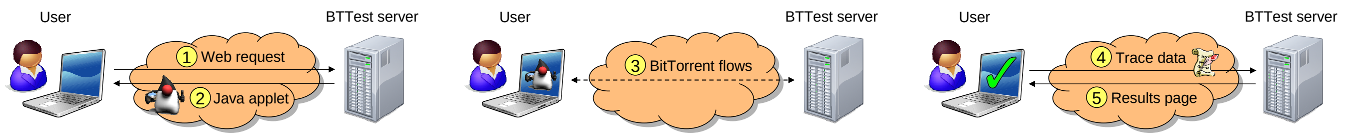


Figure 1: Overview of the BTTest system: (1) The user initiates the test. (2) The server sends her a Java applet. (3) The applet connects to the server and emulates a sequence of BitTorrent flows. (4) The applet informs the server whether any flows have been aborted. (5) The server analyzes the information from both endpoints and displays a result page.

Many ISPs are known to rate-limit the bandwidth consumed by BitTorrent traffic by deploying traffic shapers in their networks [2]. However, it has been discovered recently that some ISPs do not just rate-limit BitTorrent flows but block them outright [5] by injecting forged RST packets into the flows. When the end nodes of a BitTorrent transfer receive the RST packets, they immediately terminate the transfer.

The aggressive blocking of BitTorrent traffic by ISPs has been widely criticized, and it has generated significant interest in detecting BitTorrent traffic manipulation. While several systems have already been built to detect in-network BitTorrent blocking, they either require expert knowledge and specialized tools (which limits scalability), or they are based on high-level heuristics (which limits reliability). An example of the first category is the Electronic Frontier Foundation's 'Test Your ISP' project [4], which offers instructions for tracing a BitTorrent transfer and checking for forged packets. This method requires access to two hosts in different ISPs and involves the use of tools like Wireshark, which is beyond the capabilities of most end users. An example of the second category is the network monitor plugin for the popular Azureus BitTorrent client [8], which reports the number of aborted connections. Since the plugin does not correlate observations from both endpoints of an aborted flow, it cannot reliably determine whether the RST packets were forged or sent by the other peer.

To our knowledge, BTTest is the first tool to offer highly specific, reliable blocking detection to a large number of end users.

3. MEASUREMENT METHODOLOGY

In this section, we first present the design of BTTest and then we describe how BTTest gathers traces of BitTorrent flows.

3.1 Design goals

The goal of BTTest is to detect whether a user's BitTorrent traffic is being blocked. More specifically, we wanted to enable the user to answer the following three questions:

1. Is an ISP blocking BitTorrent flows with forged RST packets?

2. How is an ISP identifying BitTorrent flows? Is the identification based on port numbers, BitTorrent protocol messages, or both?

3. Does the blocking affect BitTorrent uploads, downloads, or both?

Note that we focus exclusively on BitTorrent blocking, and only on one specific technique, namely blocking with forged RST packets. We do *not* consider other forms of traffic manipulation, such as rate-limiting, message-dropping, or altering of the content. Detecting such a broad range of traffic manipulation practices is the subject of future work.

We wanted to deploy BTTest on a public web server and gather traces from end users around the world. Hence, another important design goal for BTTest is that it should be very easy to use. Ideally, it should be as easy to use as the test sites for measuring Internet connection speeds [9].

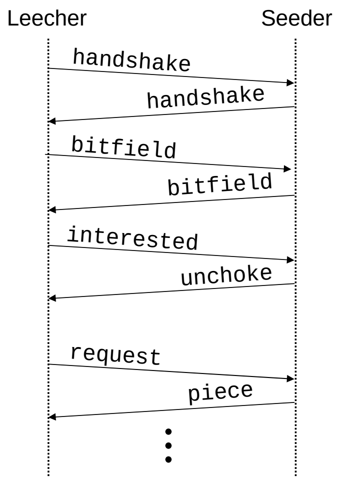


Figure 2: BitTorrent packet exchange in BTTest: The interaction always follows the same fixed script.

3.2 BTTest overview

To detect whether BitTorrent flows are blocked, BTTest emulates a series of BitTorrent flows between the user's host and a central BTTest server. During each flow, BTTest collects a packet trace, and it closely monitors both endpoints for any error conditions that might cause the flow to be aborted. If the flow is aborted without an apparent cause, BTTest checks the packet trace for additional control packets that were not sent by either of the endpoints. If such packets are found, BTTest reports this as evidence of blocking.

BTTest requires no special expertise and can be run from any machine that has a web browser with Java support. This ensures that it is available to a wide range of users. Figure 1 shows an overview of our prototype system. When a user visits the BTTest website and requests a measurement of her access link, a Java applet is downloaded to her web browser which connects[1] to our central BTTest server. This server is located in a network that is known not to block BitTorrent flows, so we can be sure that if any blocking is observed, it is performed on a link near the user's host. The applet then emulates a sequence of BitTorrent flows and reports the results back to the server. Finally, the browser displays a results page, which reports whether any blocking was observed.

3.3 Emulating BitTorrent flows

BTTest emulates BitTorrent flows between end hosts and test servers, using the standard BitTorrent protocol [6]. The interaction always follows the same fixed script, which is shown in Figure 2.

[1]To avoid problems with NAT and firewalls, the connection is always initiated by the user-side applet.

The flow can be either a downstream flow (in which data is transferred from the server to the user's host) or an upstream flow. In the following, we will refer to the sending endpoint as the *seeder*, which claims to already have all pieces of a file, and to the other endpoint as the *leecher*, which claims to have no pieces so far.

The leecher begins by exchanging a `handshake` message with the seeder. This is followed by an exchange of `bitfield` messages, which indicate the data segments that are available locally. Here, the seeder reports that it has all the segments, while the leecher reports that it has none. Next, the leecher sends an `interested` message to indicate that it wants to download segments, and the seeder grants it access by sending an `unchoke` message. During the remainder of the flow, the leecher downloads as many segments as it can; it repeatedly sends a `request` message to ask for a random segment, and the server returns a `piece` message that contains the segment. Since the content does not matter for our experiment, we fill each segment with random bytes.

3.4 BitTorrent test suite

To determine how ISPs identify BitTorrent traffic, BTTest actually runs multiple flows with different parameters. Specifically, it varies the following:

- **TCP port:** Half of the flows use port 6881, a well-known BitTorrent port. The others use port 4711, which is not associated with a specific protocol.
- **Direction:** Half of the flows transfer content downstream (from the server to the user's host), while the others transfer content upstream (from the user's host to the server).
- **Protocol:** Half of the flows contain real BitTorrent messages. The others contain messages of the same size and in the same order, but filled with random bytes.

BTTest runs each of the eight possible combinations twice, for a total of 16 test flows. Each BitTorrent flow lasts for ten[2] seconds, unless it is aborted earlier. Thus, the total number of bytes transferred depends on the available bandwidth on the path between the user's host and the server. By observing which of the tested flows are aborted, BTTest can infer how BitTorrent traffic is identified, i.e., which features actually trigger the blocking. The four test flows with random data over a non-BitTorrent port serve as a "sanity check"; they show whether the BTTest applet can communicate with our test servers at all.

3.5 Trace collection

For each emulated flow, BTTest collects two pieces of information: (1) On the server side a complete link-level packet trace (analogous to `tcpdump`), and (2) on the user side any Java exceptions the applet observed during the flow, including the point in the transfer where the connection was closed. We refer to these two items as a *result*, and to the set of all 16 results for a single host as a *result set*.

Ideally, BTTest would gather a packet trace on the user's host as well. However, there is no easy way to take such a trace from a Java applet running in a web browser, and in any case, administrator privileges (and thus a considerable amount of trust) would be required on most operating systems. Therefore, we had to find another way to determine whether the host had seen a connection reset from the server. Unfortunately, a connection reset manifests itself in Java as a generic `IOException`; the real cause is mentioned only in the string representation, which can vary between JVMs and between different languages. Our current prototype recognizes the most common strings directly and logs any other strings for further analysis.

[2]The flows are longer than strictly necessary because we also measure throughput. However, this data is not used in the present paper.

4. TRACE ANALYSIS

We now describe the analysis BTTest performs on the gathered data, and we explain the types of blocking it can detect.

4.1 Sanitizing traces

As described in Section 3.4, BTTest tries to run a sequence of 16 flows between the user's host and the server. However, some hosts abort the test early or experience problems when running the applet. Therefore, BTTest only considers a result set when the following two conditions hold:

- **All 16 flows were tested and produced a result.** Result sets which do not contain results for all 16 tests are not considered in the results below. This can be caused by the user closing her web browser or browsing to another site, or by a crash of the applet.
- **All 4 TCP "sanity check" flows were able to send some data.** Result sets where at least one of the sanity check flows had no data packet ACKed (in the case of a download) or received (in the case of an upload) are discarded. This indicates the applet was unable to contact our web server, which could be caused by misconfigured NATs, firewalls, or Java applet security policies.

If either of these conditions are not met, BTTest reports an error to the user.

4.2 Identifying blocked flows

BTTest's goal is to detect whether middleboxes in the network are inserting forged RST packets to tear down BitTorrent flows. To detect these inserted packets, BTTest analyzes the server trace along with any Java exceptions seen by the user-side applet for each flow. A flow is considered to have been torn down by a forged RST packet only when *all* of the following three conditions hold:

- **An IOException with a specific set of messages was seen by our applet.** This indicates that an error was observed with the TCP connection on the user side. BTTest looks for the messages "Connection reset by peer" or "An existing connection was forcibly closed by the remote host" in the IOException, which indicate that the host has received a RST packet.
- **The server's packet trace contains at least one incoming RST packet.** This RST packet causes the connection to be torn down at the server.
- **The server's packet trace contains no outgoing RST packets before a FIN or RST packet was received.** Once the server receives a FIN or RST packet, the connection is torn down. Thus, any subsequent data packets received on the connection will be naturally responded to with RSTs.

The presence of all three conditions strongly indicates that a forged RST caused the flow to be torn down. The first two conditions indicate that a RST was received at both the server and the user's host. While we cannot say for sure that the user's host received a RST packet (as we do not have a packet-level trace from the host), we only look for IOExceptions with messages that are caused by the receipt of a RST packet. The third condition indicates that the server did not initiate the connection tear-down (in other words, it received either a FIN or a RST before it sent any RSTs). Thus, BTTest detects forged RSTs by looking for flows (1) which were torn down by a RST received at the user's host and/or server and (2) which contain no RSTs sent by the user's host or the server before the connection was torn down.

4.3 Detecting BitTorrent blocking

We now describe how BTTest uses the information about blocked flows to detect BitTorrent blocking, and to infer how BitTorrent flows are identified by the middlebox. Our working hypothesis is that the identification could be based on three flow characteristics: the TCP port number of the flow, the BitTorrent messages in the flow, and the direction of the flow.

Recall that for each test, BTTest runs two identical flows, so it obtains two results. BTTest considers a test to have been affected by forged RSTs if either of the two flow results indicates forged RSTs. For simplicity, we call the test to have *failed* in this case; otherwise, we say that the test has *succeeded*.

BTTest then looks for BitTorrent blocking behavior by examining the result sets for each direction separately. If all tests in one direction using the BitTorrent ports fail regardless of whether BitTorrent data or random data was sent, BTTest reports *BitTorrent blocking based on BitTorrent ports* in that direction. If all the tests in one direction using the BitTorrent messages fail, regardless of the port on which the test runs, BTTest reports *BitTorrent blocking based on BitTorrent messages* in that direction.

4.4 Limitations

In its current form, BTTest can only detect a single form of traffic manipulation. It considers only BitTorrent traffic, and only blocking by injected control packets. BTTest currently does not look for traffic throttling, packet dropping, or packet manipulation. Extending BTTest to test for such additional behavior is the subject of future work.

Also, BTTest cannot determine at which point along the path the forged RST packets are generated. A typical Internet path between a host and our measurement servers is likely to cross multiple ISPs. BTTest cannot determine which ISP is responsible for tearing down BitTorrent connections. Developing techniques which use network tomography to pinpoint the location of the forged RST packets is the subject of ongoing work.

Finally, BTTest's centralized architecture makes it possible for ISPs to avoid detection by whitelisting the BTTest servers. This is unlikely to have affected the data we present in this paper, but it may become a problem once BTTest is more widely known. We are currently working on a decentralized version of BTTest, which would make whitelisting by ISPs much more difficult.

5. RESULTS

In this section, we describe how we collected a set of traces from our public BTTest server, and we present results from our analysis of these traces.

5.1 Data set

We deployed BTTest on a publicly accessible web server at *http://broadband.mpi-sws.org/transparency/bttest.php*. Initially, we invited a handful of our colleagues and friends to test their ISPs, and we asked them to spread the invitation to their friends. After the first week, the site caught the attention of a few influential bloggers, and hundreds of new users tested their ISPs each day.

From March 18th to May 7th, 2008, our BTTest servers collected a total of 47,318 result sets from end users connected to 1,987 ISPs world-wide. 146 result sets did not contain results for all 16 flows, and a further 17 failed to send data during at least one of the sanity-check flows. In these cases, BTTest reported an error to the user, so we removed these sets.

Some users ran our test multiple times. To avoid biasing our results, for each IP address, we considered only the first result set that passes the two conditions above, and we ignored all other result sets for that IP address. After removing the duplicate tests, we were left with 41,109 result sets.

We found evidence of BitTorrent blocking in 3,353 (8.2%) of the 41,109 result sets. In the rest of this section, we take a closer look at the hosts that observed blocking.

5.2 Where are the blocked hosts located?

First, we examined the countries in which hosts observed BitTorrent blocking. In total, our test was run from users in 135 countries. Most of our users came from North America (44.3%), Europe (26.7%), and South America (17.9%).

Table 1 lists all countries where we found BitTorrent blocking for at least one host. Our results indicate widespread BitTorrent blocking only for the USA and for Singapore. Interestingly, even within these countries, we observed blocking only for hosts belonging to a few ISPs.

Next, we looked at the ISPs whose hosts were affected by BitTorrent blocking. Overall, we found that hosts of 47 ISPs experienced blocking; the ISPs are listed in Table 1, along with the number of hosts we tested from each ISP and the number of hosts whose BitTorrent flows were blocked. The results show that not all hosts of these ISPs are affected by blocking.

We do not have enough data to determine why only some (but not all) hosts of an ISP are subjected to blocking, but there are several possible explanations. For example, the middleboxes that block BitTorrent transfers might not be deployed on all of an ISP's network paths, or blocking might depend on the current load of the network. Also, some ISPs might allow BitTorrent traffic up to a certain threshold and apply the blocking to the "heavy hitters" only.

5.3 How do ISPs identify BitTorrent flows?

Next, we wanted to understand what flow properties ISPs were using to detect and block BitTorrent flows. We examined each of the three flow characteristics BTTest varies in the test suite, and we determined how many of the 3,353 result sets contained evidence of blocking based on these characteristics.

- **TCP port:** We found that only 530 (15.8%) of the result sets showed evidence of blocking based on BitTorrent ports, regardless of whether or not the flows actually contained BitTorrent messages. Thus, blocking of TCP connections based only on well-known BitTorrent ports seems to exist, but does not appear to be widespread.
- **Direction:** We found that 3,335 (99.5%) of the result sets contained evidence of blocking in the upstream direction, but only 71 (2.1%) of them contained evidence of blocking in the downstream direction. Thus, ISPs seem to be blocking primarily BitTorrent uploads and are rarely interfering with BitTorrent downloads.
- **Protocol:** Finally, we found that 3,293 (98.2%) of the result sets contained evidence of blocking based on BitTorrent messages. Thus, ISPs appear to be using deep packet inspection to block BitTorrent flows regardless of the port they are using.

In summary, the BitTorrent blocking we observed seems to be focused primarily on BitTorrent uploads, and it appears to affect

Country	ISP	# measured hosts	# blocked hosts
Australia	AARNet	2	1
Belgium	MAC Telecom	5	1
Brasil	Brasil Telecom	54	1
Canada	PaeTec Comm.	9	1
	RISQ	7	1
	Westman Comm.	4	3
China	China Telecom	49	2
Finland	Joensuun Elli	1	1
Germany	Uni Göttingen	1	1
Greece	OTEnet	122	8
Hungary	DataNet	17	1
India	SonicWall	1	1
Ireland	IBIS	9	1
Jamaica	Terrenap	1	1
Kuwait	Wataniya Telecom	5	4
Malaysia	Telekom Malaysia	336	12
	Maxis Comm.	9	2
New Zealand	TelstraClear	22	1
Saudi Arabia	SaudiNet	8	1
Singapore	StarHub	156	101
South Korea	Korea Telecom	12	5
Spain	Telefonica	602	1
Taiwan	TANet	214	2
	Cheng Kung Univ.	11	2
	APOL	10	1
UK	Tiscali	354	2
USA	Comcast	4397	2574
	Cox	1004	508
	RoadRunner	2086	50
	Cablevision	646	1
	Suddenlink	123	4
	Mediacom Comm.	120	17
	Clearwire	34	9
	Midcontinent Comm.	21	13
	General Comm.	13	5
	Pavlov Media	11	2
	PaeTec Comm.	9	1
	PrairieWave	4	2
	UC Riverside	4	1
	Journey Comm.	3	1
	NHCTC	2	1
	Bergen.org	1	1
	DHL Systems Inc.	1	1
	Moric.org	1	1
	PSC	1	1
	The Shaw Group	1	1
	WSIPC	1	1

Table 1: The number of hosts with BitTorrent blocking grouped by country and ISP.

flows using the BitTorrent protocol regardless of whether or not they are using a well-known BitTorrent port.

5.3.1 *Case study: Comcast*

Our analysis found that most ISPs identify BitTorrent flows based on protocol messages. Presumably, the ISPs are using deep packet inspection to monitor the protocol messages exchanged and to decide whether a flow should be blocked. To understand the precise protocol messages that trigger blocking, we ran a controlled experiment using a Comcast host in Seattle, WA, to which we had access. In this experiment, we emulated BitTorrent transfers just as BTTest does, but we varied more aspects of the flows; for example, we obfuscated BitTorrent protocol messages by flipping bits, we left out some of the messages, and we changed the number of advertised pieces in the `bitfield` message to emulate different sharing scenarios, e.g., both peers having some but not all pieces of the file.

We found that, on this particular access link, BitTorrent uploads were blocked if and only if all of the following conditions hold:

- The server sent a valid BitTorrent `handshake` message,
- The Comcast host sent a valid `bitfield` message, and
- The Comcast host's `bitfield` message indicated that it had all pieces.

In other words, the uploads of a file were blocked only when the Comcast host has finished downloading the file and was uploading it altruistically. However, the uploads were not blocked when the Comcast host was still missing some of the pieces of the file and thus, appeared to be interested in downloading. From this experiment, we conclude that the middleboxes which tear down BitTorrent connections maintain some per-flow state and inspect the packet payload for specific protocol messages.

Note that this case study only applies to Comcast. Unfortunately, we did not have access to hosts connected to other ISPs and were therefore unable run the same controlled experiment for them.

5.4 When do ISPs block BitTorrent flows?

ISPs that have admitted to blocking BitTorrent flows claim that they do so only during the hours of peak load, when their networks are congested. The data we collected with BTTest enables us to check whether blocking occurs continuously throughout the day or is limited to just a few hours of the day. For each hour of the day, we calculated the percentage of result sets that contained evidence of blocking. For each result set, we inferred the location of the tester and then computed the local time[3] when the test had been performed. We then grouped together measurements from the same hour. Here we present data for Comcast and Cox because these are the two ISPs for which we had the most data points.

Figure 3 shows our results. While the number of measurements per hour shows a diurnal pattern with more measurements in the evening than in the early morning, the fraction of blocked tests shows no clear trend. We observed blocking for a significant fraction of the tests throughout the day. Figure 4 groups the result sets by day of the week instead. Again, there is no clear trend; we observed a significant fraction of blocked hosts on all days of the week. Finally, we used a Comcast host under our control in Seattle, WA, to run BTTest at 30-minute intervals for an entire week. We found that BitTorrent flows were constantly blocked during the entire week.

In conclusion, our data suggests that BitTorrent flows are being blocked independent of the time of the day or the day of the week.

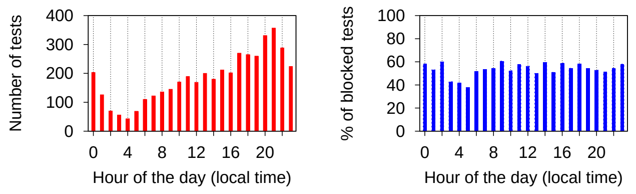
5.5 At what stage are flows blocked?

Finally, we took a closer look at the BTTest packet traces to see at which stage of the BitTorrent protocol the blocking occurred. The RST packets can be injected at different points in a transfer, that is, at different stages of the BitTorrent protocol shown in Figure 2. To perform this analysis, we used the data reported by our user-side applet about the last message it sent before the connection was torn down.

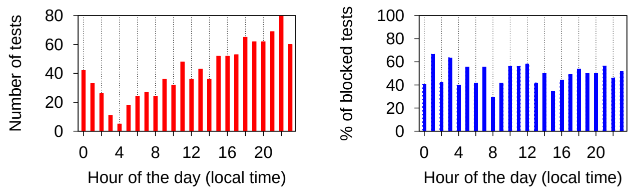
In total, we identified four different places in the protocol at which connections were blocked. We found a very strong correlation in behavior across ISPs, and we observed mostly consistent behavior for hosts of the same ISP. Due to lack of space, we only give examples for each categories.

- **After the `handshake` message:** For Telekom Malaysia and Brasil Telecom we observed that the connection with

[3]We used an IP-to-geolocation tool to infer the timezone of each tester.

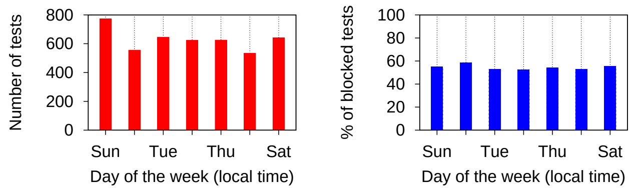


(a) Comcast (USA)

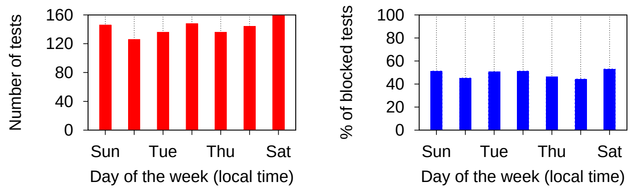


(b) Cox (USA)

Figure 3: Result sets grouped by the hour of the day for Comcast and Cox: BitTorrent flows were blocked at all times of the day.



(a) Comcast (USA)



(b) Cox (USA)

Figure 4: Result sets grouped by the day of the week for Comcast and Cox: Blocking occurred on every day of the week.

BitTorrent messages was torn down immediately after the `handshake` message was sent by the leecher.

- **After the `bitfield` message:** For StarHub, RoadRunner OTEnet, and most other ISPs we observed connection tear-down for connections with BitTorrent messages after the leecher sent the `bitfield` message.
- **After the `interested` message:** For most Comcast and Cox hosts, we observed that the connections with BitTorrent messages were torn down after the `interested` message was sent by the leecher.
- **Later in the transfer:** Finally, for Comcast, Cox and Mediacom, we observed that connections with random data on BitTorrent ports were occasionally torn down later in the transfer. However, we were unable to determine a common pattern for the exact point where the connection was torn down.

While the types of blocking can sometimes vary even between hosts of the same ISP, we found that the basic characteristics of blocking were mostly consistent across hosts and even across some of the ISPs. Because of this, we suspect that many ISPs are using similar equipment for traffic identification and reset injection, e.g., the specialized hardware sold by Sandvine [1]. However, it is possible that these boxes are configured differently in different locations or at different times of the day.

6. CONCLUSION AND FUTURE WORK

Recently published reports of access ISPs blocking BitTorrent transfers by injecting forged RST packets have sparked an international debate on network neutrality. In this context, the present paper makes two contributions. First, we presented the design of BTTest, a reliable and easy-to-use tool that allows end users to detect if BitTorrent traffic is being blocked on their access link. Second, we presented results from a large-scale measurement study that is based on a widely-used public BTTest deployment.

Our current study is limited to detecting BitTorrent blocking, and there are a number of open challenges and interesting directions for future work. First, it would be interesting to develop analysis techniques for detecting other types of traffic manipulation beyond blocking, e.g., BitTorrent traffic shaping. Second, the centralized architecture of our BTTest tool limits scalability and is vulnerable to whitelisting by ISPs wishing to avoid detection. It would be useful to investigate ways to decentralize BTTest to allow the emulated BitTorrent transfers to be sent between testing peers. Finally, while our current methodology allows us to detect BitTorrent blocking along an Internet path, we cannot diagnose where along the path the traffic is being blocked, i.e., which ISP is responsible for blocking BitTorrent. A user could potentially localize the source of blocking by repeatedly running the test from servers located at different vantage points in the Internet. By correlating the blocking data obtained from multiple transfers along different Internet paths, one could hope to deduce which links are subject to BitTorrent blocking.

7. REFERENCES

[1] “Sandvine Inc.” `http://www.sandvine.com/`.
[2] “Packeteer Inc.” `http://www.packeteer.com/`.
[3] “DslReports: Comcast is using Sandvine to manage P2P connections.” `http://www.dslreports.com/forum/r18323368-Comcast-is-using-Sandvine-to-manage-P2P-Connections`.
[4] “EFF ‘Test Your ISP’ Project.” `http://www.eff.org/testyourisp`.
[5] “Comments of Comcast Corporation before the FCC.” `http://fjallfoss.fcc.gov/prod/ecfs/retrieve.cgi?native_or_pdf=pdf&id_document=6519840991`.
[6] “The BitTorrent Protocol Specification, Version 11031.” `http://bittorrent.org/beps/bep_0003.html`.
[7] A. Parker, “The true picture of peer-to-peer file sharing.” `http://www.cachelogic.com/research/`.
[8] “Vuze Network Status Monitor.” `http://azureus.sourceforge.net/plugin_details.php?plugin=aznetmon`.
[9] “The Global Broadband Speed Test.” `http://www.speedtest.net/`.