Exploring and Improving BitTorrent Topologies

Christian Decker*, Raphael Eidenbenz†, Roger Wattenhofer‡

*ETH Zurich, Switzerland cdecker@tik.ee.ethz.ch

†ABB Corporate Research, Switzerland raphael.eidenbenz@ch.abb.com

‡Microsoft Research wattenhofer@microsoft.com

***Abstract*—BitTorrent, the most popular peer-to-peer (P2P) file-sharing protocol, accounts for a significant fraction of the traffic of the Internet. Using a novel technique, we measure live BitTorrent swarms on the Internet and confirm the conjecture that overlay networks formed by BitTorrent are not locality-aware, i.e., they include many unnecessary long distance connections. Attempts to improve the locality have failed because they require a modification of the existing protocol, or interventions by Internet service providers (ISPs). In contrast, we propose a lightweight method that improves the locality of active swarms by 6% by suggesting geographically close peers with the Peer Exchange Protocol (PEX), without any modifications to the current system. An improvement of locality not only benefits the ISPs by reducing network transit cost, it also reduces the traffic over long-distance connections, which delays the need to expand the infrastructure, easing the power consumption. We expect that if used on a large scale our method reduces the Internet's energy consumption by 8 TWh a year.**

I. Introduction

Today's Internet traffic is by and large *local*. E-mails are mostly exchanged among members of the same organization, as are VoIP calls, most popular web sites are within the same country as their users, and online gamers prefer playing on servers close-by. What about data from popular global Internet hot spots such as YouTube, Facebook or CNN.com? That is also local: replication and content caching — besides the intended performance and stability improvement — had the side effect of moving the content closer to the consumer. So for most activities on the Internet, the user will communicate only with the closest caching center, which is probably somewhere in a nearby city.

One may argue that in today's Internet, hardly any data item of significant size travels around the world more than once. Largely for this reason, the Internet infrastructure did not grow as fast as the traffic of the Internet. The last decade's main bottleneck was the last mile, and not the backbone. Aware of the locality of today's Internet, two basic questions need to be raised: is there any substantial application that is not local? And if so, can we improve on its energy efficiency by making it more local? These are the two questions addressed in this paper, and we answer both of them affirmatively.

BitTorrent is such an application as it causes massive amounts of non-local traffic. Like Karagiannis et al. [10] first pointed out, BitTorrent is anything but local; thus burdening internet service providers (ISPs) with increased transit costs. Various studies [1], [11], [15], [19], [20] attempted to quantify the traffic share that can be attributed to BitTorrent. The estimates range from 18% to 37%, depending on methodology and measurement point. Like other peer-to-peer (P2P) file-sharing systems, BitTorrent integrates the downloaders into the dissemination of the content. As such, P2P systems impose the cost of content dissemination mainly on end-users and ISPs. This is in stark contrast to classic client-server systems, where the content releaser bears most of the costs.

Numerous proposals have attempted to improve the locality of BitTorrent. However as we show in this paper, peer-to-peer file sharing is still not local. The reason for the continued lack of locality-awareness is that most of the proposals assume incentives for the end-user or the tracker operators, that are insufficient or do not exist.

Some ISPs have started to throttle or even block BitTorrent traffic [7] to counter their increased costs. Obviously, such interventions — often referred to as *traffic shaping* — are problematic as they infringe *network neutrality*, the principle that the Internet infrastructure should not prioritize certain applications at the expense of others. In this work, we introduce a method to measure the locality of live BitTorrent swarms. Additionally, we present a novel method that allows ISPs to improve the locality without interfering with the actual data transfer.

A. BitTorrent Overview

BitTorrent is a popular P2P file-sharing protocol proposed by Bram Cohen [6] in 2001. Unlike Gnutella or eMule, BitTorrent does not include mechanisms to search and discover content, instead it relies on metadata files called *torrents* to be distributed out of band. The torrent file contains the details of the content that is shared, an *info_hash* which uniquely identifies the torrent, a *tracker* that is used to establish the first contact and SHA1 hashes of the content for integrity

978-1-4799-0521-8/13/$31.00

verification.

Peers interested in sharing the content detailed in the torrent form ad-hoc networks called *swarms*. A tracker is used to bootstrap the communication. While not actively participating in the sharing of the content, a tracker keeps track of the peers that participate in the swarm. To join a swarm, a peer contacts the tracker and announces its interest to join. The tracker will then take a random subset of the peers matching the same info_hash and return them to the newly joining peer. The joining peer will then start connecting to the peers it received in the tracker response. Most clients keep a pool of about 50 connections active.

The tracker may use a variety of transports, the most common being the traditional HTTP transport and a simplified UDP transport. Recently a distributed hashtable (DHT) tracker, based on the Kademlia protocol [14], has gained popularity due to its distributed and fault-tolerant nature. Once connected to other peers in the swarm, a peer starts trading pieces. Thereby, it basically trades downloaded pieces of content for missing pieces.

There are various extensions to the original protocol including the Azureus Messaging Protocol[1], the LibTorrent extension protocol[2], protocol obfuscation and an alternative transport protocol based on UDP (μTP). The Azureus Messaging Protocol and the LibTorrent extension protocol implement a *peer exchange* (PEX) mechanism used to inform neighbors of other peers in the swarm. μTP and protocol encryption are mainly used to avoid ISP throttling or improve congestion control and do not alter the behavior of the clients.

The original BitTorrent protocol is *not* locality-aware: to enter a swarm, a peer polls a tracker for a random set of peer addresses currently active in the swarm; then the peer connects to a random set of the available peers. Note that with this process, any potential connection is established with the same probability. The yielded network topology corresponds to a random graph.

B. Related Work

Karagiannis et al. [10] were the first to recognize the optimization potential in P2P to reduce the load on inter-ISP links by improving the locality. The benefits of making the BitTorrent protocol locality-aware have been widely studied and several methods of improving the locality have been proposed. Recently, Le Blond et al. [12] provided an overview of what can be achieved if the locality is pushed to the limit in several common scenarios.

Studying locality has so far been limited to either artificial scenarios in controlled environments or partial views of the network by tapping into one or several internet service providers. Our approach is the first to measure the topology of BitTorrent swarms in the wild. Our method requires no modification to either the tracker or the actual clients participating in the swarm, nor do we actively participate in the exchange of potentially copyrighted material.

For the locality improvement there have been numerous proposals, most of which can be categorized by their method of influencing the swarm:

- modification of the client to bias the way it chooses the peers to connect to;
- modification of the tracker so that it only returns peer addresses of peers that are believed to be close to the requester;
- modification of communication between clients or tracker responses by the ISP.

TopBT [18] measures the AS distance and link hop distance by pinging a peer before establishing the connection. The selection of neighbors is then biased to prefer nearby peers. Ono [4] uses another method to identify close-by peers: based on the idea that peers that redirect to the same CDN are often close, peers give higher priority to peers with the same CDN. The information is exchanged either directly after a connection has been established or via a tracker that supports this type of localization improvement. Though not formally studied in the case of the BitTorrent protocol, Pereira et al. [16] propose keeping a variable number of connections open as to guarantee the availability of the shared data. New connections, to more distant peers, are only established if the availability is no longer granted. Again, the distance to other peers is measured in terms of routing hops, determined by the TTL field of packets. Unfortunately, while promising large improvements in locality, all of the mentioned methods either require a concerted effort of the client developers to implement the proposals; or they put additional strain on the network as active distance measurements are repeatedly conducted from multiple endpoints.

Tracker modifications on the other hand allow a single coordinating entity to improve the locality. The modified tracker announces only close-by peers to the requesting peer, thus skewing its view of the rest of the network. Biased Neighbor Selection [2] relies on the peers to inform the tracker of their location, it then uses the location information to announce only peer selections that comply with a fixed ratio of local to non-local peers. Varvello et al. [23] analyze possibilities of how the decentralized, DHT-based tracker can be used to enforce locality. They deploy *sybil nodes* that take over the region of the DHT that is responsible for queries regarding the targeted torrent. Similar to the tracker modification methods, they can announce a custom selection to each peer. Modifying the tracker allows for unilateral improvement of the locality, i.e., no modifications to the clients is required. Furthermore, tracker modifications yield large improvements up to 40%. The effect

[1] http://wiki.vuze.com/w/Azureus_messaging_protocol

[2] http://www.rasterbar.com/products/libtorrent/extension_protocol.html

is weakened if the clients use a combination of several trackers that do not coordinate the locality improving mechanisms. The effect is further weakened if the peers exchange information about available peers via PEX. Additionally, the trackers are burdened with the task of computing the best peers to suggest, although they do not have an obvious incentive to do so.

Although end users might profit from a improved locality due to reduced latency, the biggest stakeholders in BitTorrent locality are the ISPs, since they incur costly inter-ISP traffic. The simplest approach for an ISP to reduce the inter-ISP traffic caused by BitTorrent is to detect and throttle it, a behaviour which has been observed by Dischinger et al. [7]. Throttling is supposed to either disrupt BitTorrent completely or to discourage inter-ISP connections. A less intrusive method was proposed by Tian et al. [22]: ISPs should monitor swarms and replace random peer lists sent by trackers with closer peers, or they should install ISP-owned peers that act as a local cache for nearby peers. This can be seen as narrowing a peer's view of the network as it will never learn about the existence of most peers that are outside of the ISPs network. Finally, Xie et al. [24] propose a framework where ISPs and P2P applications collaborate to improve locality. All of these methods require privileged access to the transmission medium and involve the active modification of sent data. Such activity is often frowned upon and considered a violation of the network neutrality. As a result clients started encrypting the client-to-client communication and client-to-tracker communication to prevent their BitTorrent traffic from being identified and consequently dropped, or from being modified by their ISP.

While all the proposed approaches seem technically appropriate for improving locality once they are implemented, the uncomfortable question from a practical perspective is whether and how the millions of BitTorrent users or the operators of trackers could be brought to using a locality-aware BitTorrent software; unfortunately, they do not have a natural incentive to do so. Moreover, approaches that suggest ISPs to modify network messages seem unacceptable for reasons of network ethics. Approaches that require ISPs to take an active role in disseminating often copyright infringing content are even more problematic.

In stark contrast, our method does not require privileged access neither to the transmitted data nor the involved machines (clients or trackers). While we use the peer exchange mechanism for its original purpose, i.e., to widen the peer's view of the network, we also provoke locality improvements by skewing the view towards nearby peers.

II. Measuring Swarm Topologies

The sheer size of BitTorrent makes it hard to analyze: a single swarm often spans dozens of independently managed networks; peers typically run a wide variety of BitTorrent client implementations and support different protocol extensions. To understand the behavior of a swarm as a whole we have to reconstruct the topology, i.e., we have to learn how peers are connected and trade with each other. Existing methods for measuring swarm topologies typically rely on log data by the tracker, instrumenting a large number of peers or analyzing traffic logs from ISPs. These methods are inapt for measuring real world swarms, where neither the tracker nor the peers are controlled by the conductors of the experiment and the swarm is likely to span a large number of networks.

In contrast, the method we present in the following is tailored to the case of real world swarms. It enables us to detect connections between peers in a swarm, while not participating in the file sharing, and without having any particular access to the software running at the peers or the trackers. It relies solely on the BitTorrent protocol and is not limited to any particular implementation.

A. Measurement Method

When two BitTorrent peers connect for potential trading, they first exchange handshake messages. Included in such a handshake message is a 20 byte randomly generated *peer_id* that uniquely identifies the peer. The receiver of a handshake message checks whether it is already connected to a peer with the id in the message, and immediately closes the connection if so. We call the detection and successive disconnection of multiple connections a *collision*. The detection of such collisions prevents unnecessary multiple connections between peers. Duplicate connections would not increase the availability of pieces, however they would still occupy a slot in the connection pool. In experiments, we confirmed that all prevalent clients drop a new connection if it causes a collision. We exploit this sensitivity to collisions for measuring swarm topologies.

The basic functionality of our method is as follows: we repeatedly poll the tracker in a distributed manner to learn the addresses of most peers in the swarm; we then test each pair of peers, A and B, as to whether they are connected by trying to establish a connection to one of them using the id of the other, i.e., we try to connect to B using the id of A or vice-versa. If we are able to complete the handshake we can be certain that there is no connection between the peers. We therefore call a test resulting in a complete handshake a *negative result*. If on the other hand the handshake is not completed, i.e., the connection is closed before we get a handshake response, we call the test a *positive result* as it might indicate an existing connection. Unlike the case of a negative result, where it is certain that there is no connection between the peers, we must not assume that a positive result implies the existence of the corresponding connection. There are several other reasons for which a connection can be dropped without completing the

handshake. The most common ones are:

- Peers with full connection pools: the peers limit the number of connections both per torrent and globally. If the limit is reached any new connection will be dropped immediately;
- Blocking peers: a peer might block the IP address of the machine that scans the swarm, either as a result of our scanning activity or because of the fact that we use university owned IP addresses;
- Torrent removed: as soon as a torrent has been removed, a peer will start dropping connections related to that torrent, i.e., it has left the swarm;

As a first step to filter out false positives we have to distinguish dropped connections caused by our part of the handshake from connections dropped due to other reasons, i.e., connections dropped independently from any information in the handshake. This can be done by delaying our handshake by a few milliseconds. We simply ignore all tests where the connection was dropped too soon for it to be a reaction to our handshake. The delayed handshake eliminates false positives that are due to the peer blocking us, and it eliminates the false positives due to a full global connection pool. If a peer blocks from the beginning of our scan we will never get a handshake and we therefore drop all tests from that peer.

Most false positives however, are caused either by full per-torrent connection pools and by removed torrents. For both of these cases the peer needs the information from our handshake. Thus the observable behavior is identical to a true positive result. To reduce the number of false positives we reschedule positive tests and retest them again later. If the positive result was caused by an actual connection and the connection remains open for the repeated tests we will continue getting positive results for that test. The probability that a test repeatedly yields false positives when it is retested is equivalent to waiting for the first successful handshake to be completed, and therefore decreases geometrically.

Another important factor is the runtime of a scan. The runtime might be negligible in a small swarm, but the number of possible connections, and therefore the number of tests needed to detect them, grows quadratically with the number of peers in the swarm. This means that it is fairly difficult to get a consistent picture of a non-trivial swarm in its entirety, simply because the topology is likely to change during the scan. For measuring large swarms, there is a natural trade off between consistency and completeness.

In order to get an estimate for the maximum scanning duration that still allows for a mostly consistent view of the swarm we instrumented the BitThief [13] BitTorrent client to collect the connection lifetime while participating in several swarms. After filtering out connections that were not used to transfer pieces of the torrent we end up with the cumulative distribution function in Figure 1. It turns out that 85% of the connections are open for longer than 400 seconds. Unless otherwise stated all scans in our results are therefore limited to 400 seconds.

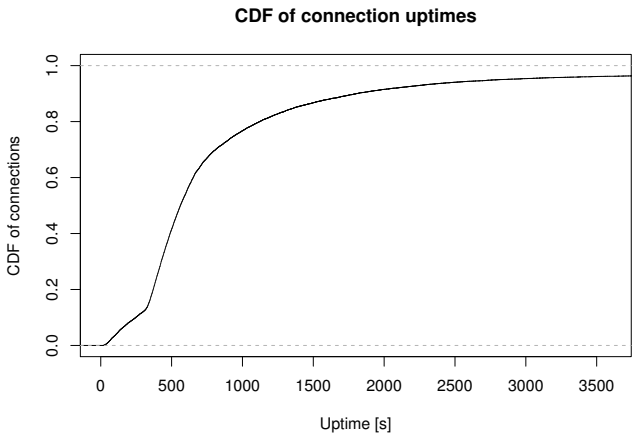

Fig. 1. Cumulative distribution function of the connection lifetime.

Finally, there are some connections that we are unable to scan. A peer might never complete a handshake with us, either because it always has a full connection pool, because it has left the swarm, or because it is blocking us. For such a peer, we cannot make any direct claim about its connections. While this is unfortunate the resulting false positives are relatively easy to filter out during the final analysis: we simply remove all positive results coming from peers that have never produced a negative result. Another type of peer that remains unscannable with our method are peers behind network address translation (NAT) or firewalls, that are not reachable from outside.

Fortunately, besides discovering the connections of a peer directly, we can also discover connections incident to a peer when scanning for its peer_id at the other peers in the swarm. Indirect scanning not only allows us to detect connections of peers that cannot be scanned directly, it also helps eliminate false positives since each connection is tested at both endpoints. To discover the peer_ids of peers behind a NAT or firewall we listen for incoming connections. If a NATed peer connects to our scanner, we learn its peer_id, store it and include it in the scan.

B. Implementation

Our measurement method tests each connection individually. Consequently, scanning an entire swarm with n peers requires $O(n^2)$ tests. In addition to the $n^2 - n$ unique tests we reschedule all positive scans to be checked again later. Luckily, tests can be run concurrently. More precisely tests at different peers can be executed in parallel, to reduce the time complexity to $O(n)$. To distribute the load on several machines

we split the scanner into three parts:

- A *coordinator* that collects peer_ids and peer addresses as they are discovered in the swarm and schedules tests;
- A number of *testers* that execute tests as assigned by the coordinator;
- A *log collector* that collects the results from the executed tests.

The coordinator initiates the scanning process by loading a torrent file, parsing its contents and distributing it to the testers. The testers are processes that run on PlanetLab [5] nodes. They open a port and listen for incoming connections for each torrent they get issued, poll the trackers for peers and receive scan tests from the coordinator. A scan test consists of the handshake information and the address of the peer to be tested. Upon receiving a scan test the tester opens a new connection to the address in the scan task and attempts to carry out a handshake.

Tests may result in the handshake being completed, the handshake not being completed, or in a connection failure, when the connection could not be established. Connection failures occur either because of a time-out or because the connection is rejected.

The coordinator schedules at most one test for each tester and each tested peer. If we were to schedule multiple tests originating from the same IP address at once we would introduce more false positives; the reason being that a peer accepts only one concurrent connection from each IP address. The second connection attempt would be rejected. As we execute the tests from several PlanetLab nodes we may schedule multiple concurrent scan tasks with the same target peer since the tests do not originate from the same IP address.

Let us denote the maximum number of concurrent scan tasks targeting the same peer as *concurrency*. The concurrency is thus a parameter of our measurement method. A higher concurrency increases the probability of provoking a false positive that is due to the fact that our testers occupy too many slots in the connection pool of a peer.

The log collector has a feedback channel to the coordinator to inform it about positive tests that are subject to rescheduling. Additionally, the log collector notifies the coordinator about peers that have caused connection failures. Such peers are given low priority since they are unlikely ever to be reachable.

C. Evaluation

To evaluate the performance of our method we start with a bird's eye view of the swarms and discuss the issue of what part of them is observable in a best case; then we evaluate our measurement method as to how well it explores the observable part; finally, we discuss the precision of our findings.

An average of 42.24% of the IPs returned by the trackers were reachable during our tests, the remaining peers either rejected connections or timed out. 57.76% of the peers in the swarm are therefore either behind a NAT or a firewall, or they left the swarm before we contacted them.

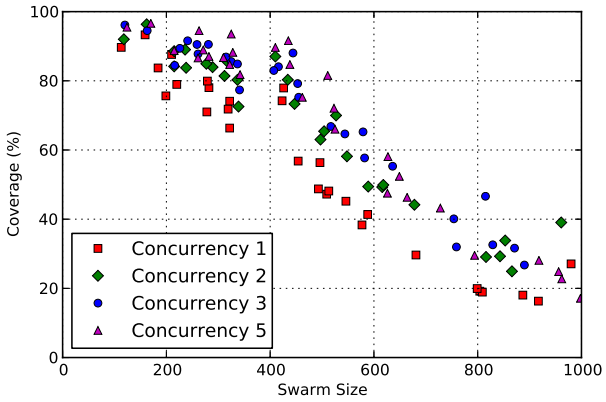

Fig. 2. Network coverage by swarm size for various concurrency settings

Assuming that connections between NATed peers are just as likely as between non-NATed peers we would only be able to detect $100\% - 57.76\%^2 \approx 66.63\%$ of the connections in a swarm. We argue however that establishing a connection between two NATed peers is far less likely.

The only option for two firewalled peers to establish a connection is by using NAT-holepunching [8]. As NAT-holepunching requires additional coordination and direct connections are much easier to set up, we assume that we detect a much larger part of the network and we consider 66.63% a very conservative a lower bound.

To estimate the percentage of the network that we are able to explore we ran several scans and compared the coverage to the swarm size that we were able to achieve. To calculate the coverage we used the following formula:

$$coverage = \frac{|tests|}{(|peer\_ids| - 1) \cdot |peers|}$$

Where $tests$ is the set of scan tasks that have been executed, $peer\_ids$ is the set of peer_ids and $peers$ is the set of peers that are reachable. The reason we differentiate between peers and peer_ids is that peers behind a NAT that provided us with their peer_id are not directly scannable. To see if a higher concurrency allows for a higher coverage we ran several scans with varying concurrency levels and varying swarm sizes. Figure 2 plots the resulting coverages for concurrencies of 1, 2, 3 and 5.

As one might expect, the coverage of the network decreases linearly with the size of the scanned network, despite the number of tests growing quadratically. For large swarms, churn is the main reason for the low coverage as peers leaving the swarm make parts of the search space unscannable and newly

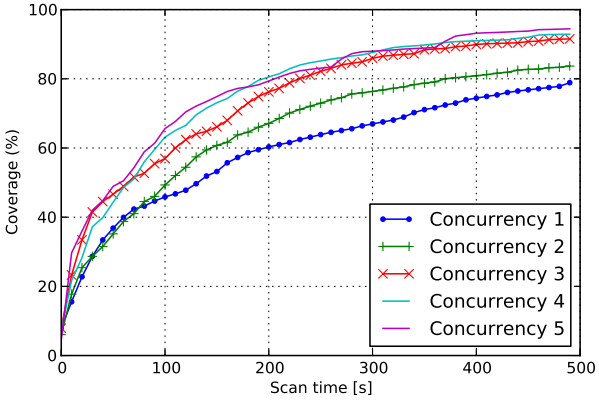

Fig. 3. Temporal evolution of the coverage in a swarm with 250 peers for different concurrency levels.

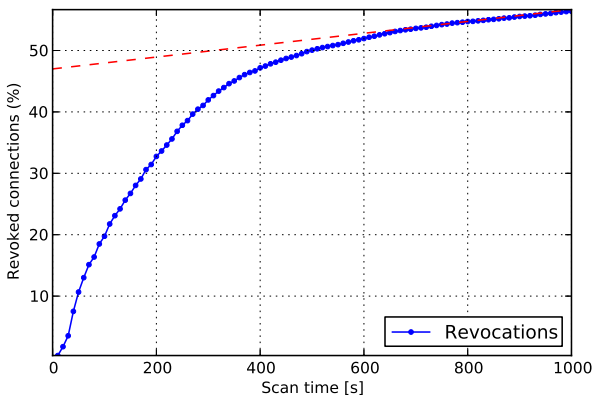

Fig. 4. Ratio of revoked connections in relation to the scan time duration. The swarm scanned for this purpose had 303 distinct peer_ids.

joining peers extend the search space. Peers joining at a later time will be added to the search space but the scanner will have less time to complete the scan on those peers. Additionally, peers greatly vary in response times, which we define as the time from a task being scheduled to its result arriving to the log collector. Some peers have an average response time close to 30 seconds. With such peers we can test only 13 peer_ids in 400 seconds.

Note that increasing the concurrency does not have a large effect on the coverage. It turns out that the reason for the relatively small effect is that during a measurement, the coverage quickly converges to its final level, and is not improved significantly anymore. To corroborate the convergence this we ran 5 scans with different concurrency levels. We then parsed the logs with increasing time frames to simulate shorter scans. Figure 3 plots the coverage as a function of the scanning duration. The coverage for each of the shorter scans was calculated using the peer_id count at the end of the 500 second scan to get monotonically increasing coverages. It therefore includes also peer_ids that are yet unknown in the shorter scans. For higher concurrency levels the scan converges faster towards the final coverage, but the performance gain when using a concurrency of 5 instead of 3 is not as pronounced as when using a concurrency of 2 instead of 1.

Finally we have to quantify the errors introduced by false positives. To detect false positives we rescheduled positive scan tasks to be tested again later. While retesting allows us to eliminate false positives, it also causes us to misclassify connections that are active during the first test but are closed before the second test. To quantify the amount of legitimate connections that we lose during the retesting, and how effective the retesting is in identifying false positives we ran a scan for a longer period of 1000 seconds. We then parsed the log with increasing time frames to simulate shorter scans. We ended up with 100 scans with 10 second increments each. For each scan we calculated the number of connections (tests that produced a positive at first) and the number of revoked connections (tests that resulted in a positive but were negative during the retesting). See Figure 4 for a plot of the ratio of revoked connections to the detected connections.

We observe that after 400 seconds the revocation rate of detected connections becomes more or less linear. In this linear part of the plot the revocation rate due to false positives is negligible; the increase in revoked connections is mostly due to legitimate connections being closed, which causes later tests to produce negatives. Hence we can extrapolate the number of legitimate connections in the swarm at the beginning of the scanning process. In this particular swarm the ratio of revoked connections after 1000 seconds was 0.563. By assuming that a linear amount of good connections is revoked by retesting we can compute a theoretical false positive rate of 0.505 at 400 seconds, of which our scan detected 0.479. This leads us to the conclusion that only 5.15% of the detected connections in this particular swarm are false positives. Note that for smaller swarms the revocation rate converges faster, and most of the scanning time is spent on retesting positive results.

D. Locality Unawareness

Our scanning technique allows us to reconstruct large parts of the topology of a BitTorrent swarm. What we end up with is an undirected graph with connections as edges and the peers, identified by their peer_id and their IP address, as vertices. Figure 5 illustrates the combination of all scanned swarms, plotting each peer in its approximate geographic location and connecting them if they are connected in the swarm.

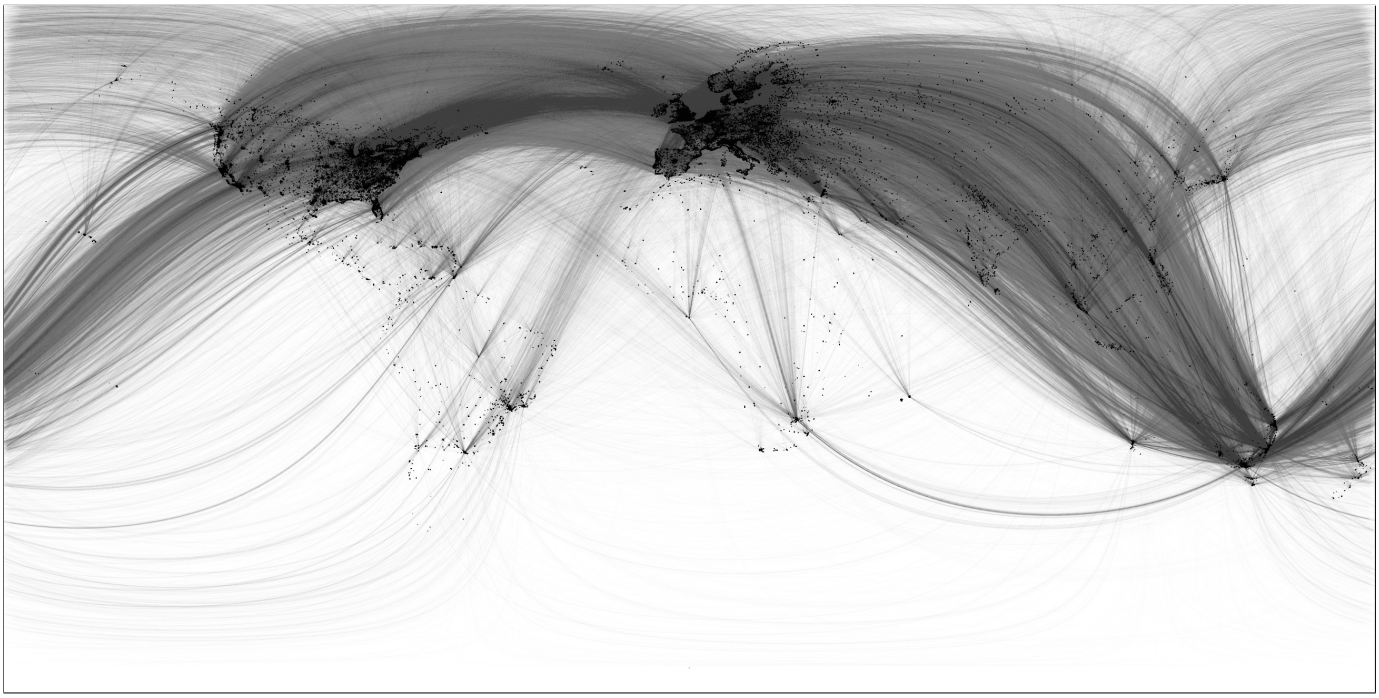

Fig. 5. A composite rendering of 1.9 million peers and 14.4 million connections from several swarms. The swarms were selected from ThePirateBay.se Top 100 List. The connections trace the great circle distance, i.e., the shortest distance on the earths curvature between the peers. Multiple connections are indicated by the opacity of the line.

Definition of Locality: Let $d(a, b)$ denote the distance between peers a and b; let $\sigma(a, b)$ be the indicator function that equals 1 if a and b are connected and 0 otherwise; and let D denote the random variable that corresponds to the distance $d(a, b)$ between a pair (a, b) that is chosen uniformly at random. Thus, the locality $\mathcal{L}$ is given by

$$\mathcal{L} := \frac{\sum_{a,b} \sigma(a, b) \cdot d(a, b)}{\sum_{a,b} \sigma(a, b)} \cdot \mathbb{E}[D]^{-1}.$$

The intuition is that if the connections are chosen independently from the distance measure d then the average connection length has to be close to the expected distance value for randomly chosen connections. The expected distance $\mathbb{E}[D]$ can be calculated as the average connection length of all possible connections, i.e., the distance between all possible pairs (a, b). If $\mathcal{L}$ is close to 1, this indicates that the swarm is not locality-aware. If $\mathcal{L} > 1$ distances are even longer than if connections were chosen uniformly at random. The smaller $\mathcal{L}$ is, the better the locality.

It is worth noticing that false positives and coverage do not influence the locality. The coverage merely dictates the size of the sample, whereas the false positives act as noise that slightly skews the result towards the expected value.

In absence of the details of the underlying physical network topology we resort to using the geographical great circle distance of the endpoints to estimate the actual distance in the underlying physical network topology. Huffaker et al. [9] showed that there is a strong correlation between geographical great circle distance and the underlying network topology, be it Autonomous System (AS) path length or IP path length. If there is any large scale use of locality improving strategies present in the BitTorrent networks, it should considerably lower the average AS path length or IP path length as compared to random connections, as well as the average geographic distance between peers in a swarm.

In our measurements, we use the MaxMind IP Geolocation[3] library to retrieve the geographical coordinates from peers' IP addresses, and compute the distances between peers from those coordinates. Together with our scanning method, which provides the value of $\sigma(a, b)$ for two peers a and b, we have a tool that gives a good approximation of the locality $\mathcal{L}$ of a given swarm.

In a series of measurements, we scanned and computed the locality of 33 swarms that had between 50 and 500 active peers. These swarms were chosen randomly among the most recently published 1000 torrents of the two torrent discovery sites with highest Alexa rank, *thepiratebay.se* and *kat.ph*.

We observed that most swarms have a locality close to 1, and the average locality over all measured swarms is 1.062. This result strongly suggests that there are no locality improving measures present in today's BitTorrent networks. Arguably, the existing proposals have not been adopted due to the weak incentives.

[3]http://www.maxmind.com

III. Improving Swarm Locality

In this section we describe a method to influence the topology of a BitTorrent swarm by suggesting nearby neighbors. We argue that existing methods to improve the locality have not been widely adopted because they rely on weak incentives for the end-users and the tracker operators. Moreover ISPs, which would benefit from an improved locality of BitTorrent swarms, have not had at their disposal the tools necessary to improve the locality. Our method does not rely on privileged access to either the software or the transmission medium, instead it relies on an existing protocol extension to extend the peers view of the network and skew it to include more nearby peers.

A. Suggesting Neighbors

As developers of BitTorrent software cannot be forced to implement locality improving measures, we take a different approach and improve the locality of swarms externally, by suggesting geographically close neighbors to peers with *peer exchange* (PEX) messages. PEX is a widely adopted extension to the BitTorrent protocol that enables peers to send each other lists of peer addresses. Thus, once connected to some peers, more peers can be discovered using PEX without polling the tracker. According to our tests over 93% of the peers support PEX.

While we have no power to dictate which connections a peer should open, we can skew its set of known peers to include more nearby neighbors. The peer will randomly select a peer from the set of known peers for a connection attempt. It is therefore more likely that the resulting connection is to a nearby peer, instead of a random long connection. A PEX message contains the contact information, i.e., the IP addresses and ports, of other peers in the swarm. Each peer information contained in a PEX message is called a *suggestion*.

A suggestion is called *successful* if it results in a real connection between the two peers. Note that the success probability of a single suggestion depends on the size of the receiver's connection pool, the number of already connected peers, and the number of peer addresses known from other sources (trackers or other peers).

A single PEX message may contain several suggestions. The number of peers in a PEX message, though not limited in the protocol specification, is typically limited to 50 peer addresses by the implementations.[4] A peer receiving too many suggestions in a PEX message might assume that the sender is misbehaving and discard the PEX message. We therefore limit the number of suggestions in a single PEX message to 50.

Let p be the pool size at the receiving peer; C the set of established connections; N_k the set of known peers, excluding established connections, and N_s the set of peers suggested in a PEX message, excluding established connections. We define the random variable X that counts the number of connections established as a result of a suggestion. The probability that at least one suggested connection is established as result of the PEX message is given by the hypergeometric distribution:

$$P[X \geq 1] = 1 - P[X = 0] = 1 - \frac{\binom{|N_s|}{0} \cdot \binom{|N_k \setminus N_s|}{p-|C|}}{\binom{|N_s \cup N_k|}{p-|C|}}$$

In order to maximize the probability of at least one successful suggestion for a PEX message we should try to maximize the number $|N_s|$ of suggested peers, maximize the number $p - |C|$ of free slots in the connection pool, and minimize the number $|N_k|$ of otherwise known peers that are reachable. To maximize $|N_s|$ we always send the maximum allowed number of peers in the PEX message. Both of the latter conditions cannot be changed by the suggesting party, but can be observed in peers that have only recently joined the swarm.

A recently joined peer has just polled the tracker for peers in the swarm and is starting to connect to peers until its connection pool is full. It is therefore desirable to identify recently joined peers as early as possible and suggest nearby neighbors as they are most likely to open a connection as the result of such a suggestion.

In order to make suggestions we first need to create a local view of the swarm in which the peers of the swarm are stored along with their coordinates. To create the local view we can either repeatedly poll the tracker [21] or in the case of an ISP, the peers can be directly extracted from the monitored traffic. Unlike modifications or shaping of traffic, the monitoring of traffic does not infringe network neutrality. The same two mechanisms can also be used to identify newly joining peers.

As soon as a new peer is identified it is inserted into the local view of the swarm. We then compute a selection of peers that are reachable and close to the joining peer based on our view. Once the closest peers, i.e., the ones that will be suggested, have been identified a new connection to the targeted peer is initiated and the PEX message is delivered.

To maximize the impact of our suggestions we always suggest at both ends of a desired connection. Note that, similar to NATed peers, we can even influence a peer that does not support PEX by suggesting it to the desired neighbors. Additionally the regular contact with known peers serves the purpose of keeping the local view up-to-date. This includes detecting peers that leave the swarm, so we do not suggest them in the future.

Our method of suggesting nearby peers via PEX can be considered a non-intrusive alternative to other external methods that influence the swarm topology. Unlike other methods which try to restrict a peer's view to a subset of the peers in the swarm, our method amplifies its view and skews

[4] http://wiki.theory.org/BitTorrentPeerExchangeConventions

it so that selected peers are more likely to be close-by.

B. Implementation

Similar to the implementation of the measurement method we divide the suggester implementation into two separate parts:

- A *coordinator* that keeps track of swarm membership, identifies joining peers and creates suggestions;
- Several *suggesters* that poll the tracker, report the tracker responses to the coordinator and carry out suggestion tasks that they receive from the coordinator.

The suggesters are processes running on several PlanetLab nodes. Once started they connect to the coordinator which will issue either a tracker poll request or suggestions to be delivered.

To initialize the local view of the swarm the coordinator issues a tracker poll request to all suggesters. The peer addresses received from the tracker are then mapped to their geographical location using the MaxMind GeoIP library. To allow efficient querying for nearby peers the geographical coordinates can be arranged on a space filling curve. The distance on the space filling curve over the coordinates approximates the great circle distance between the coordinates.

After the local view has been created we can start suggesting nearby neighbors to joining peers. To identify joining peers as quickly as possible, the suggesters are scheduled to continuously poll the trackers at short intervals. When a joining peer is detected it is inserted into the local view. Then the 50 peers closest to the new peer are computed from the local view, and the suggestion task is assigned to one of the suggesters.

The suggester opens a new connection to the peer that is to receive the suggestions, completes the handshake, sends a PEX message containing the suggestions, and closes the connection again. Should the suggester be unable to contact a peer and cannot deliver the PEX message it notifies the coordinator. The coordinator in turn removes the peer from the local view.

As mentioned, besides sending suggestions to the joining peers we also suggest them to the other end of the desired connections. The suggestions to already known peers are delayed until the suggestions to the joining peers have completed. Delaying the suggestions allows multiple suggestions to the same host to be grouped while the peers that are going to be suggested are tested for reachability. Thus we reduce the overhead and ensure that the suggested peers are reachable.

C. Evaluation

To evaluate the effectiveness of our PEX suggestion method, we measured a total of 79 swarms chosen randomly from the torrents recently uploaded to *thepiratebay.se* and *kat.ph*.

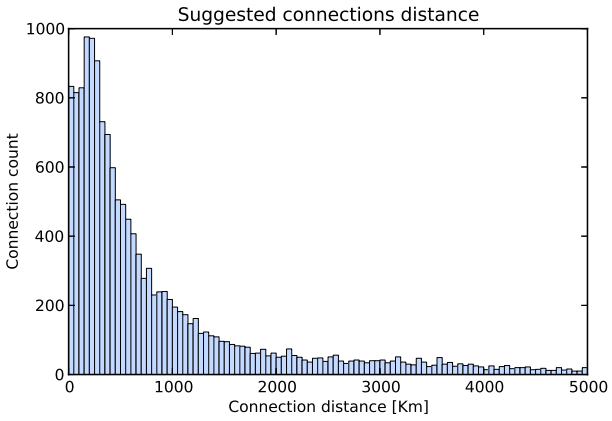

Fig. 6. Histogram of the average distance between the peer receiving the suggestion and the suggested peers.

Recently uploaded torrents are more likely to have a large percentage of newly joining peers for which our method is particularly effective. Our results should also be applicable to longer running swarms with more peers, assuming that the suggesting has already started early during the creation of the swarm. Out of all swarms, we influenced half as described and left the other half unaffected for comparison. As the suggestions take some time to take effect, we let the swarm run for one hour, influenced or not, and then measure its locality with the method described in Section II. We influenced a total of 37 swarms by sending 27'599 PEX messages to newly joined peers, each with 50 suggestions. Figure 6 shows the distribution of the average distance between the suggested peers and the receiver. On average the suggestions had a connection length of 867.90 km whereas the average connection length over all swarms was 6796.50 km. In 755 cases we were able to suggest all 50 peers in the same AS as the joining peer, i.e., all 50 peers were in the same ISP's network.

To measure the effective locality improvement we used our measurement method described in Section II. Figure 7 compares the locality of individual connections in influenced swarms and non-influenced swarms. The suggestions increase the probability of very short connections: the average rate of connections shorter than 1000 km is 31% in swarms that have been influenced, while only 24% of connections are shorter than 1000 km in swarms that have not been influenced. Overall the influenced swarms exhibit a locality of 0.994 on average while the non-influenced swarms have an average locality of 1.062. Thus, our PEX suggestion method achieves an average improvement of the locality of 6.3%.

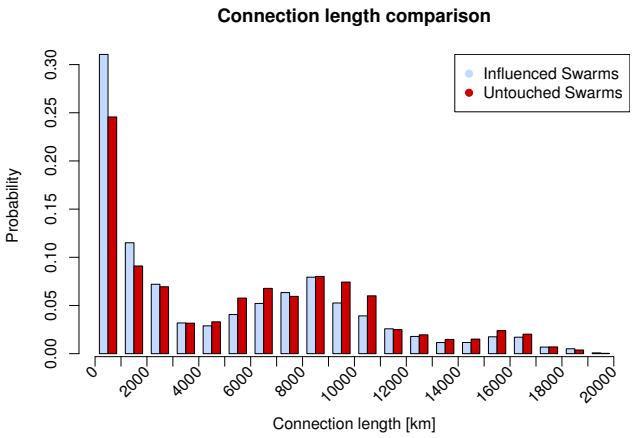

Fig. 7. Comparison of connection lengths between influenced swarms and untouched swarms.

IV. Conclusion

Using a novel technique we were able to infer connections between peers in BitTorrent swarms. By applying our method at a large scale we reconstructed the topology of some swarms and verified that swarms are all but local. We then proposed a lightweight method to improve the locality of a swarm by suggesting close by peers. This increases the probability of creating short distance connections, while not imposing our decision to the client. This method can be implemented by the ISPs, which we argue is the party with the largest incentive to do so.

An improved locality may also have an impact on the energy consumption of the Internet infrastructure. Chabarek et al. [3] showed that routers are not *power-proportional*, i.e., the power consumption is not proportional to the traffic they route. But a reduced load on long distance backbones delays the need for capacity upgrades, saving both the embodied energy of the devices as well as the active energy consumption for running them. According to Rhagavan and Ma [17] the Internet infrastructure, i.e., routers, telecom switches, fiber optics and copper cables, account for 139.4 TWh/year in electrical energy consumption and in 542.5 TWh/year in embodied energy. The assumption that the energy cost of a transmission is linear to the distance it has to travel allows us to estimate the impact of our method. With a conservative estimate of 18% for the traffic share, BitTorrent can be said to be responsible for 27.9 TWh/year in active energy consumption and 108.5 TWh/year in embodied energy.

If applied at a large scale, our method could save up to 1.7 TWh/year in electricity and up to 8.0 TWh/year in embodied energy. It could therefore save energy equivalent to the power consumption of a city like Boston.

References

[1] The Application Usage and Risk Report. Technical Report June, Palo Alto Networks, 2012.

[2] R Bindal, P Cao, W Chan, J Medved, G Suwala, T Bates, and A Zhang. Improving traffic locality in BitTorrent via biased neighbor selection. In *Proc. of the IEEE International Conference on Distributed Computing Systems*, 2006.

[3] J. Chabarek, J. Sommers, P. Barford, C. Estan, D. Tsiang, and S. Wright. Power awareness in network design and routing. In *Proc. of the IEEE International Conference on Computer Communications*, 2008.

[4] D R Choffnes and F E Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. In *ACM SIGCOMM Computer Communication Review*, 2008.

[5] B Chun, D Culler, T Roscoe, and A Bavier. Planetlab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 2003.

[6] B Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, 2003.

[7] M Dischinger, A Mislove, A Haeberlen, and K P Gummadi. Detecting bittorrent blocking. In *Proc. of ACM SIGCOMM conference on Internet measurement*, 2008.

[8] B Ford, P Srisuresh, and D Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference*, 2005.

[9] B Huffaker, M Fomenkov, and D Plummer. Distance metrics in the Internet. *Proc. of IEEE International Telecommunications Symposium*, 2002.

[10] T Karagiannis, P Rodriguez, and K Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *Proc. of the ACM SIGCOMM conference on Internet Measurement*, 2005.

[11] C Labovitz, S Iekel-Johnson, D McPherson, J Oberheide, and F Jahanian. Internet inter-domain traffic. In *Proc. of the ACM SIGCOMM Conference*, 2010.

[12] S Le Blond, A Legout, and W Dabbous. Pushing BitTorrent locality to the limit. *Computer Networks*, 55(3), 2011.

[13] T Locher, P Moor, S Schmid, and R Wattenhofer. Free riding in BitTorrent is cheap. In *Proc. of the Workshop on Hot Topics in Networks*, 2006.

[14] P Maymounkov and D Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Proc. of the International Workshop on Peer-to-Peer Systems*, 2002.

[15] J S Otto, M A Sánchez, D R Choffnes, F E Bustamante, and G Siganos. On blind mice and the elephant: understanding the network impact of a large distributed system. In *Proc. of the ACM SIGCOMM Conference*, 2011.

[16] R L Pereira, T Vazão, and R Rodrigues. Adaptive search radius-lowering internet p2p file-sharing traffic through self-restraint. In *Proc. of the IEEE International Symposium on Network Computing and Applications*, 2007.

[17] B Raghavan and J Ma. The energy and emergy of the internet. In *Proc. of the ACM Workshop on Hot Topics in Networks*, 2011.

[18] S Ren, E Tan, T Luo, S Chen, L Guo, and X Zhang. TopBT: A Topology-Aware and Infrastructure-Independent BitTorrent Client. In *Proc. of the IEEE International Conference on Computer Communications*, 2010.

[19] Sandvine. Global Internet Phenomena Report: Spring 2011. Technical report, Sandvine, 2011.

[20] Klaus Mochalski Schulze Hendrik. Internet Study 2008/2009.

[21] M Sirivianos, J H Park, R Chen, and X Yang. Free-riding in BitTorrent networks with the large view exploit. In *Proc. of the International workshop on Peer-To-Peer Systems*, 2007.

[22] Chen Tian, Xue Liu, Hongbo Jiang, Wenyu Liu, and Yi Wang. Improving BitTorrent Traffic Performance by Exploiting Geographic Locality. In *IEEE Global Telecommunications Conference 2008*.

[23] Matteo Varvello and Moritz Steiner. Traffic localization for DHT-based BitTorrent networks. *Proc. of the International IFIP TC6 Conference on Networking 2011*.

[24] H Xie, Y R Yang, A Krishnamurthy, Y G Liu, and A Silberschatz. P4p: provider portal for applications. In *ACM SIGCOMM Computer Communication Review*, 2008.