This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE P2P 2011 proceedings

B-Tracker: Improving Load Balancing and Efficiency in Distributed P2P Trackers

Fabio V. Hecht, Thomas Bocek, Burkhard Stiller
University of Zurich, Institute of Informatics (IFI)
Zurich, Switzerland
Email: {hecht,bocek,stiller}@ifi.uzh.ch

***Abstract*—Trackers are used in peer-to-peer (P2P) networks for provider discovery, that is, mapping resources to potential providers. Centralized trackers, *e.g.*, as in the original BitTorrent protocol, do not benefit from P2P properties, such as no single point of failure, scalability, and load balancing. Decentralized mechanisms have thus been proposed, based on distributed hash tables (DHTs) and gossiping, such as BitTorrent's Peer Exchange (PEX). While DHT-based trackers suffer from load balancing problems, gossip-based ones cannot deliver new mappings quickly. This paper presents B-Tracker, a fully-distributed, pull-based tracker. B-Tracker extends DHT functionality by distributing the tracker load among all providers in a swarm. Bloom filters are used to avoid redundant mappings to be transmitted. This results in the important properties of load balancing and scalability, while adding the ability for peers to fetch new mappings instantly. B-Tracker shows, through simulations, improved load balancing and better efficiency when compared to pure DHTs and PEX.**

I. Introduction

Trackers are important building blocks used in peer-to-peer (P2P) systems for provider discovery – mapping *resources*, such as files or video segments, to *providers*, that is, peers that announce the ability to provide them. In the simplest case, *e.g.* the original BitTorrent protocol [1], the tracker follows a client/server (C/S) approach.

C/S-based trackers, however, do not fully benefit from P2P properties, such as no single point of failure, scalability, load balancing, and the lack of a central authority. Therefore, different types of distributed trackers have been deployed. Distributed hash tables (DHTs) are natural candidates to be used as distributed trackers [2], [3], since their main functionality is mapping keys (content) into values (providers). Another way of designing a distributed tracker is using a gossip protocol, such as Peer Exchange (PEX) [4], [5], enabling peers to spread information about potential providers directly to each other.

This paper introduces B-Tracker (Balanced Tracker), a fully-decentralized, pull-based tracker that improves both efficiency and load balancing of a DHT-based tracker. Using B-Tracker, each provider becomes itself a tracker for the resources it provides. Mechanisms for tracker discovery and updating information are defined. In addition, Bloom filters [6] are used to avoid peers discovering already known providers.

B-Tracker may be used by any P2P application that employs a tracker to locate possible providers, for instance file-sharing applications, such as BitTorrent. Delay-sensitive applications, *e.g.* live streaming, may further benefit from its pull approach by requesting peers when those are needed.

Evaluations show that the proposed approach achieves better efficiency and load balancing when compared to a pure DHT approach and PEX.

The remainder of this paper is organized as follows. Section II presents related work. The suggested approach is described in Section III. Evaluation details and results are presented in Section IV. Section V contains conclusions and suggests future work.

II. Related Work

Two types of distributed trackers, namely DHT and gossiping, have been proposed and deployed, such that trackers are as well able to benefit from P2P properties.

Distributed hash tables (DHTs), such as KAD [3], are able to map *keys* (*e.g.*, content) into *values* (*e.g.*, providers). The DHT functionality is modified to allow several values to be added to a single key and to return a random subset of those values when queried. DHT-based trackers are pull-based, which allows a peer to retrieve a new set of providers as soon as and only as long as it is needed. The fact that a random subset is returned, regardless of which providers the requester has already obtained, reduces its efficiency, since a large amount of traffic may be used to transfer useless providers, and newly arrived providers with free resources will not be as quickly discovered by peers in the swarm. Another problem is load balancing; since content popularity resembles a power-law distribution [7] and DHTs keep a constant number of replicas per key, peers responsible for popular content have much higher load than others.

Provider information may also be spread using a gossip protocol, such as Peer Exchange (PEX) [4], which is implemented by several BitTorrent clients as an extension of the original protocol. Using PEX does not eliminate the need for a tracker, since every peer must still contact a tracker (C/S or DHT) in order to obtain its first provider list. Though different implementations of PEX exist, their main idea is that peers keep their neighbors informed about their current neighbor set. This is done by periodically (*e.g.*, once a minute) sending messages containing sets of added and removed neighbors [5] to every neighbor. When new providers are needed, peers select providers that appear least frequently as their neighbors' neighbors, since those are probably newly

978-1-4577-0149-8/11/$26.00 ©2011 IEEE

TABLE I
RELATED WORK COMPARISON

Approach	*Efficiency*	*Load Balancing*	*Push/pull*
DHT	-	-	Pull
PEX	-	+	Push
B-Tracker	+	+	Pull

arrived ones that might have free resources. PEX reduces mean download time [8] and improves load balancing in BitTorrent, due to every peer being responsible for sending regular update messages. But, owing to its push-based approach, a trade-off on the frequency of messages sent must be considered. If sent less frequently, the information is spread more slowly, which may be troublesome, especially for delay-sensitive applications, such as video streaming. If sent more frequently, efficiency decreases, as information will be more redundant.

Table I displays a comparison between the expected efficiency and load balancing properties of DHT, PEX and B-Tracker. Efficiency refers to the traffic generated to spread the knowledge about providers, while load balancing refers to how well traffic is distributed among peers.

III. B-TRACKER

Though the terms *peer*, *tracker*, *provider*, and *neighbor* all refer to a participant in the P2P system, this terminology defines more precisely the different roles that peers perform in different situations. In short, a *peer* queries a *tracker* to obtain a list of *providers*, which are contacted directly and, if there is mutual interest, may become *neighbors*, with which actual resource provisioning takes place. The basic functions a tracker offers to peers are *getProviders(resourceID)*, which returns a list of providers of the resource, and *addAsProvider(resourceID)*, which adds the sender of the message to the provider list at the trackers. Finally, *removeAsProvider(resourceID)* is called by a peer that is not anymore a provider for the given resource. It is assumed that, once a peer is provided with a resource, *e.g.* a file or a video stream, it becomes itself a provider of it – a *seeder* or a *leecher* in BitTorrent jargon.

A. Primary Trackers

B-Tracker uses a DHT structure for initial tracker discovery, since DHTs offer a scalable structure to store key-value mappings at well-known locations. Peers with *peerID* closest to the key (the *resourceID*) are responsible for storing the list of providers of the resource. These peers – termed *primary trackers* – are discovered in $O(\log n)$ steps, where n is the number of peers in the system. The DHT's original *put(key, value)* function is modified to allow multiple tuples (*peerID, IP address, TCP or UDP port* of providers) to be stored under a single key, and *get(key)* is adapted to return a random subset of these tuples.

The number of primary trackers for a resource is given by the primary tracker replication factor r_p. Since primary trackers are not necessarily providers for the resources they track, and to motivate peers to use secondary trackers, they have limited provider storage capacity, holding only up to c_p providers per resource.

B. Secondary Trackers

Once a peer has obtained a provider list from a primary tracker, subsequent tracker queries can be issued to any provider, since each of them is a *secondary tracker* for the resources they provide. The concept of secondary trackers improves scalability, since resources with more providers are able to distribute the load among more trackers, and fairness, because the load is shared by those peers interested in providing the resource.

An important parameter is n_p, the number of primary trackers that peers query before requesting from a secondary tracker. While a low value decreases the load of primary trackers, it may return peers that have none or outdated information about secondary trackers. A high value reduces this risk, but increases the number of requests to primary trackers. Secondary trackers are not limited in storage capacity, since they scale with swarm popularity.

C. Improving Efficiency

In order not to suffer from the Coupon Collector Problem [9], and to avoid that each tracker keeps state about which providers were supplied to each request, a solution based on Bloom filters [6] is used. Queries to primary and secondary trackers include a Bloom filter with all already known providers. Trackers take the filter into consideration by returning a random subset of known providers for the specified resource, excluding providers that match the filter. While Bloom filters save bandwidth due to their fixed size, they may produce false positives. This means that an unknown provider may not be found. The probability of false positives can nevertheless be adjusted to a low value.

D. Updating Mechanism

The *addAsProvider(resourceID)* operation is used by peers that have become providers for a certain resource. It can be issued to both primary and secondary trackers, but primary trackers accept only up to c_p providers per resource.

The replication factor r_p determines on how many primary trackers the operation is attempted, while r_s represents the number of replicas at secondary trackers. A higher r_s value increases the chance that a provider is found, but increases communication costs. A random subset of known secondary trackers is chosen to receive *addAsProvider(resourceID)*, as a simple way to distribute the load evenly among them.

E. Outdated Information

Tracker entries tend to become outdated as peers fail, leave, or stop offering resources, without informing the responsible trackers with a *removeAsProvider(resourceID)* message. This is a problem for all centralized or distributed tracker approaches. Having trackers periodically verify all providers they hold would bring a large overhead due to the potentially

large number of entries. B-Tracker assigns a time to live (TTL) to each resource-provider mapping stored on primary and secondary trackers. Providers are required to update (via *addAsProvider(resourceID)*) their respective tracker entries before the TTL runs out.

IV. Evaluation

B-Tracker has been implemented and evaluated using simulations to show its properties, when compared to popularly deployed distributed tracker approaches, namely PEX and pure DHT-based trackers. TomP2P [10] has been adapted to support DHT, PEX, and B-Tracker approaches.

The evaluation focuses on efficiency and load balancing. Load is defined in terms of upstream traffic, since it is a scarce resource in a P2P system. Efficiency is defined in terms of mean load per peer in the swarm, considering all tracker-related messages sent, so less load conveys better efficiency. Load balancing is defined as the standard deviation of load among all peers in the swarm, so less deviation determines better load balancing.

The parameters used for the evaluation are as follows. The Bloom filter assumes a probability of false positives $p = 0.0073$ with a number of items $n = 100$, which results in a filter of size $m = 1024$ bits [6]. A fixed replication factor $r_p = 20$ is used for the DHT approach, as in the popular BitTorrent implementation [3]. B-Tracker uses $r_p = 2$ and $r_s = 18$ for replication, since they add up to 20, in order to be fairly comparable to the DHT approach. The number of primary trackers that peers consult before requesting from a secondary tracker $n_p = 0$, that is, they always query secondary trackers for providers first, resorting to primary trackers only if all queries to secondary trackers fail. Primary tracker storage capacity $c_p = 35$ providers per resource, since 35 is a common number of neighbors used by P2P applications. All results are averages from 100 runs.

A. Simulation Setup

A P2P system with 1000 peers was simulated as follows. At each run, a swarm is initially created with 50, 250, or 450 peers. Peers in the swarm are interested in obtaining a certain resource, *e.g.*, downloading a file. Each peer in the swarm obtains 35 providers from the DHT. Measurement starts only after they have obtained those initial providers, in order to simulate a live swarm. The system, then, suffers from churn, which is defined as the percentage (10%, 20%, 30%, or 40%) of peers in the swarm that go offline, being immediately substituted by the same number of newly created peers. All peers attempt in turn to have again 35 providers in total, exchanging messages according to the approach in place.

In the DHT approach, peers query always a random one of the 20 peers that are responsible for holding the provider list for the resource in question. The tracker always replies with a random subset of at most 35 providers.

In the PEX approach, peers exchange PEX messages containing a set with newly added neighbors and a set of disconnected neighbors. If, after exchanging PEX messages, a peer still does not have 35 providers, it queries the DHT to obtain them.

In the B-Tracker-NF – NF stands for *no Bloom filters* – approach, peers query one or more random secondary trackers, which in essence are providers obtained from the initial DHT call until they obtain at least 35 providers. If, after querying all known secondary trackers, a peer has not reached its goal, it queries a primary tracker to obtain them. The B-Tracker approach works like B-Tracker-NF, except that all requests contain a Bloom filter holding the currently known providers.

B. Evaluation: Efficiency

In a more efficient system, the knowledge of which are current providers is spread with less traffic generated per peer. Efficiency is, therefore, defined in terms of the average load per peer; load being defined as bytes sent per peer, on average. Figs. 1, 2, and 3 show the average load per peer for swarm sizes 50, 250, and 450, respectively. Each value shown is an average of all runs, with error bars displaying the standard deviation.

B-Tracker achieves better efficiency when compared to pure DHT and PEX approaches. A DHT approach is not efficient because each new peer and peers with less than 35 providers in the swarm need to query the DHT, which creates many routing messages to find trackers. PEX is even less efficient, because it requires that peers send many unnecessary messages, informing neighbors about their new neighbors regardless of whether or not it is needed. B-Tracker shows better efficiency than B-Tracker-NF due to the use of Bloom filters – though request messages are larger, since they contain the filter, the provider list returned by trackers contains only useful information, further improving overall efficiency.

The B-Tracker approach shows better scalability, since, for larger swarms, the mean load per peer increases only slightly. DHT and PEX experience a larger load increase from swarm size 50 to 250, though from 250 to 450 it increases only slightly. The difference between B-Tracker-NF and B-Tracker shows that using Bloom filters as proposed improves efficiency especially on larger swarm sizes.

Load also increases with churn for all investigated approaches, since, with more churn, there are more newly created peers that look for providers, and more peers need to obtain more providers. PEX, however, has a higher load increase with higher churn when compared to the other approaches.

C. Evaluation: Load Balancing

Figs. 4, 5, and 6 show the standard deviation in load among all peers in the swarm. Load balancing was calculated for each run and an average for all runs is displayed; error bars show, thus, the standard deviation for the different runs.

B-Tracker-NF and B-Tracker distribute load much better than DHT, due to the presence of secondary trackers. In a pure DHT approach, peers that are trackers become heavily loaded as swarm size increases, as seen in Fig. 6. PEX shows improved load balancing, especially on larger swarms. B-Tracker-NF shows that using Bloom filters as proposed

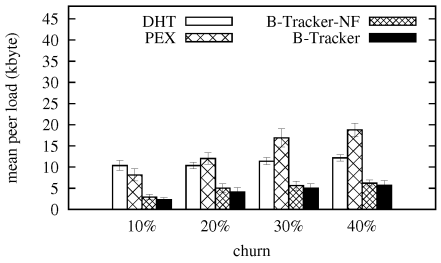

Fig. 1. Efficiency, swarm size 50

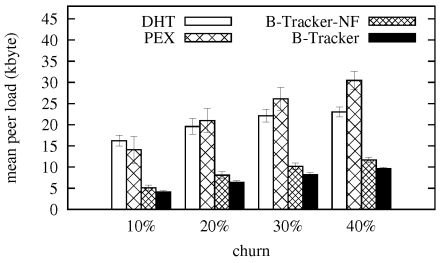

Fig. 2. Efficiency, swarm size 250

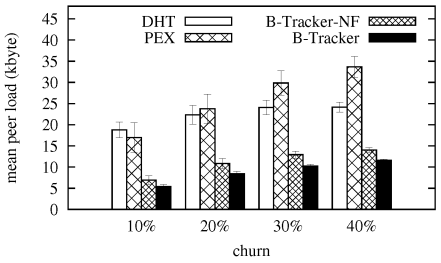

Fig. 3. Efficiency, swarm size 450

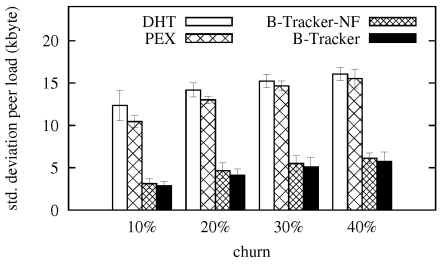

Fig. 4. Load balancing, swarm size 50

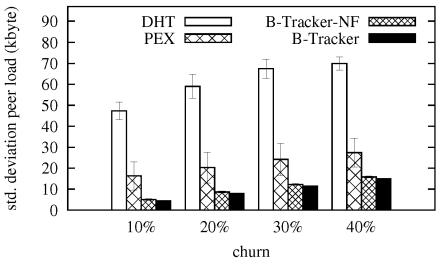

Fig. 5. Load balancing, swarm size 250

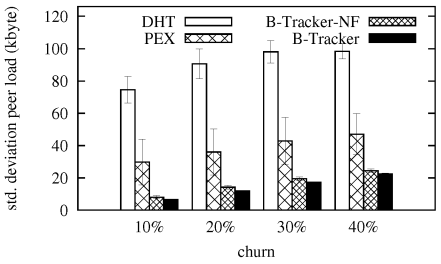

Fig. 6. Load balancing, swarm size 450

improves the load balancing only by a small amount. The fact that load balancing degrades on larger swarms on all approaches is explained by their use of a DHT for initial tracker discovery, besides as a last resort if PEX and secondary trackers do not yield the goal of 35 providers per peer.

Churn has a negative influence on load balancing in all investigated approaches and swarm sizes. This is also due to the DHT being queried at least initially by all new peers. The difference of these load balancing steps, however, is small between 30% and 40% churn rates, suggesting that it increases at smaller steps.

V. Conclusions and Future Work

This paper introduces B-Tracker, a pull-based, fully-distributed P2P tracker. B-Tracker improves load balancing by increasing the number of replicas proportionally to content popularity. Its pull approach allow the use of Bloom filters to eliminate irrelevant providers from tracker replies, and eliminates providers being sent to peers which are not interested in receiving new providers.

Simulations show that B-Tracker achieves better load-balancing and higher efficiency than other distributed trackers. A pure DHT approach shows poor load balancing because it uses a fixed replication factor. PEX shows improved load balancing but lower efficiency, since peers exchange messages which may not be of interest. Finally, a larger swarm size and higher churn produce only small degradation in B-Tracker's both load balancing and efficiency. The use of Bloom filters as suggested helps a further increase in system efficiency by avoiding redundant traffic.

Future work will investigate security aspects with malicious peers and trackers, establish theoretical bounds for B-Tracker operations, investigate locality-awareness of secondary trackers, and deploy B-Tracker on a real P2P system.

Acknowledgment

This work has been performed partially in the framework of the EU ICT STREP SmoothIT (FP7-2008-ICT-216259) and EU CSA SESERV (FP7-CSA-2010-258138).

References

[1] B. Cohen, "Incentives Build Robustness in BitTorrent," in *1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[2] M. Steiner, T. En-Najjary, and E. W. Biersack, "A Global View of KAD," in *7th ACM SIGCOMM conference on Internet measurement*, ser. IMC '07. New York, NY, USA: ACM, 2007, pp. 117–122.

[3] S. A. Crosby and D. S. Wallach, "An Analysis of BitTorrent's Two Kademlia-Based DHTs," Department of Computer Science, Rice University, Tech. Rep. TR-07-04, June 2007.

[4] "Peer Exchange," http://wiki.vuze.com/w/Peer_Exchange, last visited: March 2011.

[5] "BitTorrentPeerExchangeConventions - TheoryOrg," http://wiki.theory.org/BitTorrentPeerExchangeConventions, last visited: April 2011.

[6] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[7] F. Hecht, T. Bocek, and D. Hausheer, "The Pirate Bay 2008-12 Dataset," http://www.csg.uzh.ch/publications/data/piratebay/, December 2008.

[8] D. Wu, P. Dhungel, X. Hei, C. Zhang, and K. Ross, "Understanding peer exchange in bittorrent systems," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, aug. 2010, pp. 1 –8.

[9] V. G. Papanicolaou, G. E. Kokolakis, and S. Boneh, "Asymptotics for the random coupon collector problem," *Journal of Computational and Applied Mathematics*, vol. 93, no. 2, pp. 95 – 105, 1998.

[10] "TomP2P Project Site," http://tomp2p.net, last visited: April 2011.