Designs and Evaluation of a Tracker in P2P Networks

Adele Lu Jia and Dah Ming Chiu
Department of Information Engineering
The Chinese University of Hong Kong
{jl007, dmchiu}@ie.cuhk.edu.hk

Abstract

The "tracker" of a P2P system is used to lookup which peers hold (or partially hold) a given object. There are various designs for the tracker function, from a single-server tracker, to multiple-server tracker system, to DHT-based serverless systems. In this paper, we classify the different designs, discuss the different considerations for these designs, and simple models to evaluate the reliability of these designs.

1 Introduction

In Peer-to-Peer (P2P) systems, the term *tracker* originated from the design of the popular file sharing system BitTorrent (BT) [2]. A tracker is a server that is used to bootstrap a P2P system, an otherwise entirely distributed system. The most critical function provided by a tracker is to *introduce* other peers engaged in the same activity to a requesting peer. In order to perform this function, a tracker keeps track of peers as soon as they make a request. A tracker may also perform other related peer management functions. For example, peers may be required to periodically report to the tracker for keeping other statistics. Furthermore, a tracker can also be used to authenticate peers before providing them any service.

Subsequently, many other P2P content distribution systems, including many P2P streaming and Video-on-Demand (VoD) systems such as [3], adopted similar architectures as BitTorrent. All these P2P content distribution systems divide the content into many pieces and distribute them through different dynamically formed peer trees based on what pieces of content different peers are holding. This approach is referred to as *data-driven*, or *unstructured* P2P content distribution algorithms. Its surprisingly good performance and adaptability in the face of peer churn and heterogeneous resource availability is attracting a lot of interest in the research community. This is particularly the case for the distributed algorithms for forming the peer overlay (peer selection algorithms in BT's terminology) and for scheduling the piece exchange among peers (piece selection algorithm in BT's terminology).

In this paper, we focus on the design of the tracker function, which is a key enabler in this architecture. We use the word *tracker* to mean the service (provided by a BT tracker) rather than the server itself. There are two main approaches in tracker design, one is based on using deployed servers (we call server-based approach), and the other is based on using the peers themselves (we call peer-based approach). Based on these two approaches, there are many variants in the design, supporting scalability in the number of objects (e.g. files) and scalability in the number of peers simultaneously accessing the same object. For example, distributed hash table (DHT)[6][4] and replication may be applied in tracker design. Our contribution is to provide a systematic description of the different designs of the tracker function, give a general discuss of the pros and cons of the different approaches, and present some simple models for comparing the reliability of these designs.

The paper is organized as follows. In section 2, we provide a taxonomy of different tracker designs. In section 3, we discuss the merit of different designs. In section 4, we present some simple models to compare the reliability of different designs quantitatively, and derive some general observations from these models and their system parameters. Finally, we give conclusions and future directions.

2 A taxonomy of tracker designs

For the data-driven P2P content distribution architecture, it is necessary for each peer to discover other peers engaged in the same content distribution session, as well as what pieces of content these peers have. The tracker usually only supports the discovery of peers. The discovery of what pieces peers hold is normally accomplished by *gossip*[2], in other words, by a peer directly querying its neighboring peers. What pieces peers hold changes frequently with time, so in most scenarios only gossip can provide the most timely information without overburdening a server and in-

curring excessive network overheads. While it is possible, broadly speaking, to consider this (providing piece information) also part of the tracker function, we take a narrower view. That is, the tracker only maps an object (distribution activity) to a set of peers (partially) holding this object.

Therefore, the tracker needs to deal with only two kinds of information: (a) objects, and (b) peers; and provide the mapping between them. Objects are the files (in P2P file sharing or VoD) or video channels (in P2P streaming). A tracker should be able to serve multiple objects simultaneously. Peers are the users downloading the objects. Each peer registers with the tracker for the object it is downloading; and requests for a set of other peers downloading the same object. Although a broader view of the tracker can include additional interactions between the tracker and the peers (e.g. statistics collection), we assume the minimum responsibility for the tracker in this study.

Tracker design can be classified by the following three dimensions:

Who provide the tracker function? There are basically two choices: using *deployed servers* (**DS**), or using peers (**P**). In the latter case, it is possible to rely on only a subset of the more powerful peers known as supernodes.

How are objects assigned to tracker nodes? In the same P2P system, there may be many objects made available for sharing. Instead of having one tracker node serving all these objects, multiple tracker nodes (whether DS or peers) can share the load. The assignment can be by manual configuration (**M**), or via a distributed hash table (**DHT**).

How are peers assigned to tracker nodes? A large number of peers may be accessing the same object simultaneously, causing too much load for a single tracker node to handle the load. There may be other locality and reliability reasons for having multiple tracker nodes serve a single object. In this case, the assignment depends on whether the tracker nodes are deployed servers or peers. In the former case, the assignment can be based on user choice, if tracker nodes are explicitly advertised to users (**U**), or can be automatic (**A**), if the tracker node must be found by a DHT mechanism. In the latter case, the assignment has to be automatic (**A**).

Let us now consider some examples of these different designs below.

In the classic BitTorrent, the tracker is a server and the binding of the tracker node to the object is advertised in a meta-file (the "torrent" file)[2]. A user (peer) can choose the tracker based on which meta-file it selects to use (or a specific tracker in a meta-file with multiple trackers). The peer then contacts the tracker to find other peers downloading the same object. We can refer to this design as (DS+M+U). **For scalability, BitTorrent also enables DHT recently to perform tracker function. Being backups for servers, BitTorrent's DHT function is similar to that of eMule, another popular P2P file downloading protocol.**

eMule connects to two networks at the same time, ED2K and KAD. In KAD network, eMule uses DHT to let the tracker function be shared by peers themselves. It uses a particular DHT algorithm known as Kademlia [4][1]. The basic idea of any DHT algorithm is that it provides a mapping from an object name to a *target node* that keeps some information about the object of interest. In reaching this node, the lookup process may have to traverse several intermediate nodes. A well-designed DHT also provides some redundancy (via replication) in the paths reaching any object. The mapping from the object to the set of trackers for the object is then stored at the target node. In our taxonomy, this design can be labeled (P+DHT+A). **In ED2K, tracker function is performed by deployed servers, similar to that in BitTorrent.**

A third example is the PPLive VoD system. According to the designers [3], the tracker function is provided by several deployed servers, and a DHT is used to allocate the objects (video files) to this set of servers. This design can be labeled as (DS+DHT+A).

3 Design considerations

In designing the tracker function, there are many considerations. Many of them are not quantifiable. We discuss them briefly here.

Ease of implementation: A simple client server model should be simpler than DHT, and this can be the reason for the original tracker design.

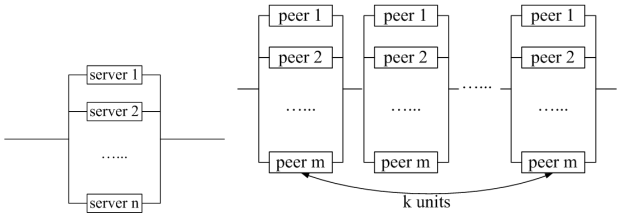
Legal liability or management responsibility: There may be legal liability in running a tracker. It also incurs management chores. So a serverless (based on DHT) design is very desirable.

Costs: There are also some costs associated with running a tracker, e.g. the server and bandwidth costs. With a serverless tracker, these costs are absorbed by the peers.

Flexibility: Implementing tracker in servers certainly gains more control for the content distributor (in the case when content comes from on distributor rather than from the peers themselves). For example, the content provider may make peers in different networks/countries use different trackers and form different sessions.

Security: On the one hand, server-based tracker can be subjected to DoS attacks, and DHT-based tracker can be more robust in that regard. On the other hand, server-based tracker can be used to implement some access control policies.

While the above considerations are all important and could decide the tracker design, another important consideration is the reliability which directly affects user perception. This metric can be quantitatively evaluated, by system models presented in the next section.



(a) Server-based (b) DHT-based

Figure 1. Reliability system model

4 Performance Models of Tracker Design

We measure reliability by a simple criterion: *probability of lookup success*. This probability depends on two factors: (a) Are the tracker node(s) required to answer the lookup request up and running? (b) Are the tracker node(s) required to answer the lookup request overloaded? The first factor can be evaluated based on standard reliability theory[5]. The second factor can be evaluated based on simple queueing theory.

4.1 Reliability

Let us first assume the tracker nodes all have infinite capacity, so performance is not a problem.

For reliability, the tracker design can be modeled either as a *parallel* set of units, or a *series* set of units where each unit may be composed of a parallel set of units, as shown in Fig. 1. The former captures the design of manually-configured, server-based tracker with some degree of replication (DS+M+U); whereas the latter captures the DHT-based tracker design (P+DHT+A). The lookup process in the DHT-based tracker traverses a path of tracker nodes (served by peers), where each tracker node may have several other peers serving as backups.

Let R_{server} and R_{DHT} denote the reliability of the two tracker designs; and let R_s and R_p denote the reliability of deployed servers and peers respectively. Then:

$$R_{server} = 1 - (1 - R_s)^n, \tag{1}$$

where n is the number of deployed nodes serving as the tracker for one object.

$$R_{DHT} = (1 - (1 - R_p)^m)^k, \tag{2}$$

where k is the expected path length of tracker nodes visited by the DHT, and m is the number of backup peers for each tracker node.

Each term, R_s or R_p, can be further expressed in terms of the *mean time to failure* and *mean time to repair* for the two kinds of tracker nodes; also, different tracker nodes may all take different reliability values. Furthermore, the expected path length k may depend on the peer population size (e.g. log of population size is an upper bound). These details can be added without changing the nature of the above models.

4.2 The Performance Factor

In reality, both deployed servers or peers have finite capacity, and they can be overwhelmed under heavy load. So the node reliability R (which can be R_s or R_p) can be viewed as the product of the following factors:

$$R = P_{up} P_{queued} P_{served} \tag{3}$$

The first term, P_{up}, takes the original value R_s or R_p respectively. The last term, P_{served}, the probability that the tracker node stays up till the request is served, is normally very close to 1, and can be ignored. The second term, P_{queued}, the probability that the tracker node is not overwhelmed, can be derived from a simple queueing model, e.g. an M/M/1/h queue where h is the queue length (the number of simultaneous lookup request accepted by the tracker node). The request service rates, μ_s or μ_p, are properties of the tracker nodes. The request arrival rate, λ, is given by the workload of the model, derived from the peer arrival rate and the average number of objects each peer accesses while in the system. Based on these parameters, we can derive the reliability of a server-based tracker as follows:

$$R_s = P_s * (1 - \frac{\rho_s^h - \rho_s^{h+1}}{1 - \rho_s^{h+1}}) \tag{4}$$

where

$$\rho_s = \frac{\lambda}{n\mu_s}$$

assuming the requests are evenly distributed to the n server nodes.

The node reliability for a peer node, R_p, can be written down in a similar fashion. There is a significant difference: since there are more peers in the system serving as tracker nodes, the request arrival rate to each peer, ρ_p, would be much lower than the request rate to a tracker node in a server-based system. The exact formula depends on the particular DHT algorithm, and the computation of average peer population in the system, and is not include here due to space limits. The bottom line is that since $\rho_p \ll 1$ there is little reliability effect due to blocking, and $R_p \approx P_p$.

Finally, R_s and R_p can be plugged into the reliability formula to derive the lookup success rate of both server-based or DHT-based trackers.

5 Observations from Modeling

The tracker models can be used to evaluate tracker design alternatives for various network and workload parameters. They can also be further refined to serve as capacity

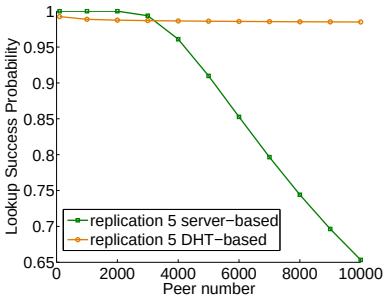


Figure 2. Influence of peer number

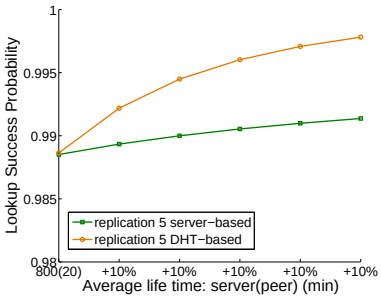


Figure 3. Influence of churn

planning tools for deploying the tracker function to meeting reliability and performance objectives. For example, we have done some preliminary work in extending the models to include the repair process. While it is more straightforward for the deployed server case, the DHT case is more complicated since it also relies on a periodic maintenance mechanism. Due to space limitations, we have not included the extended models in this paper.

Based on the simple models described in the last section, we can make some general observations:

The reliability of server-based tracker design can deteriorate quickly under heavy load, whereas the DHT-based tracker can scale with load since when the load increases the number of servers will increase automatically, see Fig. 2. This problem can be remedied by careful capacity planning.

For both the server-based and DHT-based trackers, the increase in mean-time-to-failure of the tracker node can improve the tracker reliability, as expected. But it is interesting to see the sensitivity to this parameter is stronger in the DHT-based design, see Fig. 3. The reason is that our model assumes that a server needs a random repair time before becoming available again.

For both the server-based and DHT-based trackers, the increase in replication improves reliability, as expected. In this case, the sensitivity is about the same for both designs, see Fig. 4.

Besides reliability, some other interesting dimensions to quantify the tracker performance also exist, e.g. tracker load for different rates of access and performance cost of locating content. We simply contain the former in our queuing model and the latter is a new metric to compare different tracker designs. All of these improvements and extensions will be included in our future work.

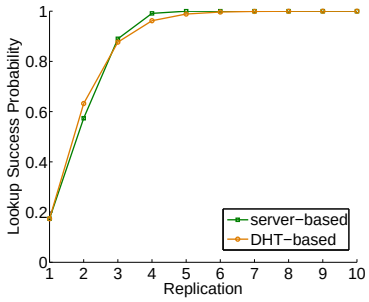


Figure 4. Influence of replication

6 Conclusions

In this paper, we provide a preliminary study of the design of tracker in p2p systems and its evaluation. More detailed modeling and analysis is in progress. We are also interested in building a capacity planning tool for server-based tracker design, which requires more detailed modeling of various other functions of a tracker not included in our simple models.

References

[1] D. Carra and E. Biersack. Building a reliable p2p system out of unreliable p2p clients: The case of kad. In *Conext 2007*, Hong Kong, 2007.

[2] B. Cohen. Incentives build robustness in bittorrent. In *P2P Econornics Workshop*, Berkeley, CA, 2003.

[3] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p vod system. In *to appear in ACM Sigcomm 2008*.

[4] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, March 2002.

[5] M. Rausand and A. Hoyland. *System Reliability Theory: Models,Statistical Methods, and Applications*. John Wiley & Sons, 2004.

[6] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Sigcomm'01*, San Diego, California, USA, August 2001.