Rabit: A Reputation Architecture for BitTorrent

Hani Ragab-Hassen, Olga Jones, Nikos Galanis
School of Computing
University of Kent, UK
E-mail: {h.ragab,oj24,nk228}@kent.ac.uk

***Abstract*—With the proliferation of the use of peer-to-peer networks for distributing multimedia and software content, there is an increasing need for efficient and reliable data distribution mechanisms that can scale to large numbers of users. The peer-to-peer protocol BitTorrent is scalable and is widely used to share content. The lack of reputation techniques in BitTorrent makes it impossible to get any information about how reliable a peer is.**

In this paper, we propose a secure and reliable architecture for reputation in BitTorrent. Our framework allows a peer to securely share its feedback about another peer after a transaction. Then, allow other peers to securely retrieve this information when required. Furthermore, we discuss how secure our system is by studying how it protects against the well knows peer-to-peer attacks.

***Index Terms*—Peer-to-peer, BitTorrent, reputation, trust, distributed hash tables.**

I. Introduction

Peer-to-peer systems take advantage of the resources available at end-users. A pure P2P system is self-organised and does not require any central administration [1]. Peers equally share the tasks of maintaining the system function. The main distinction of peer-to-peer (P2P) networking and the traditional client/server model is how the network is structured. The client/server model is a centralized system with the server fulfilling clients requests. The limitations on server resources might lead to bottlenecks. P2P systems avoid this problem by distributing the system management and operations load across the peers, thus shifting the network load to the end-users. The most widely known application of P2P is file sharing.

BitTorrent is one of the most popular P2P networks. In BitTorrent, the information about a shared file is stored in a *.torrent* file [2]. This file contains, among other details, the IP address of a *tracker*. The main role of a tracker is to maintain a list of the peers currently downloading a particular file. The set of peers downloading/uploading the same file is called a swarm. Once the peer has the .torrent file (e.g. from a public search engine), it extracts the tracker's address and contacts it. The tracker replies with a list of the peers currently in the swarm, and the peer can contact them and start downloading the file. A peer that has the whole file is called a *seeder*, any peer that is still downloading the file is called a *leecher*.

As described above, BitTorrent is centralised because of the need of the tracker. In its recent implementations, BitTorrent includes Distributed Hash Tables (DHT) mechanisms [3] that allow to run a trackerless version of BitTorrent, thus rendering the system decentralised. The trackless torrents are quite different because there are no trackers and peers find each other using their local DHT tables. There are several implementations of DHTs, Kademlia is the DHT implementation used in BitTorrent clients. Clearly, the new version of BitTorrent is closer to pure P2P systems (even if it still relies on the trackers to initialise the DHTs). We propose our architecture for this version of BitTorrent.

The reputation of a peer is as a collective measure of trustworthiness [4]. It is based on votes (ratings) from peers who interacted with that peer in the past. The reputation information could be used to provide a differential quality of service to the participating peers.

Free riding is a common problem in P2P networks [5], [6]. A free rider has a selfish behaviour and tries to take the maximum of resources they can while sharing no or minimum resources. A typical example of free riders in BitTorrent is a peer that leave the swarm as soon as they finish downloading the file.

The Bittorent's tit-for-tat mechanism called choking algorithm was proposed to avoid free riding. The peer decides who to upload the pieces of the file to in favour of the peers providing the highest uploading rate at the moment. The term unchoked is used to describe the peers who are allowed to download from the client. The rest of the peers are chocked. The clients recalculates every few seconds which peers to unchock. Peers are allowed to unchock some new peers regardless of their uploading rate, this allows the new peers to download few pieces of the file to start with, then the new peer can share them [7].

Locher et al. showed in [8] the tit-for-tat mechanism can easily be subverted with the free BitTorrent client BitThief [8]. Moreover, the altruistic peers such as the seeders are not rewarded for the free sharing of their resources. The download reward resulting from using tit-for-tat, even in an unmodified client, does not go beyond one download. We therefore are searching for an incentive mechanism to encourage peers to share the resources. Such mechanisms can be based on the peer's reputation. The incentive mechanism would be expressed in giving a higher quality of service to peers with a higher reputation score. There are two main problems in P2P networks that the reputation systems can be used for; the first is bad files, ranging from low quality files to malware infected files. The second is the free riding problem. We focus on the latter problem.

In this paper, we propose a secure and scalable reputation framework that does not require making major changes to the existing BitTorrent protocol. The peers reputation scores produced by our scheme will be used to provide differential quality of service that depends on

978-1-4673-0921-9/12/$31.00 ©2012 IEEE

Authorized licensed use limited to: UDESC Universidade do Estado de Santa Catarina. Downloaded on September 08,2021 at 17:58:40 UTC from IEEE Xplore. Restrictions apply.

the scores. In section II we review some P2P reputation schemes. Section III describes our architecture Rabit that we discuss in section IV. Section V concludes our paper.

II. Related works

Although the reputation systems design for P2P networks is a relatively new area of research, several reputation schemes exist in the litterature. Hoffman et al. in [9] analyse reputation systems according to three criteria : the formulation, calculation and dissemination of the information. Formulation deals with how to collect the feedbacks from peers. Calculation specifies how the reputation scores or the trust values are calculated from the feedback, as well as where the calculation is performed and how the results are stored. Dissemination addresses the area of how the results of the score calculations are communicated to the peers. Only few reputation schemes address the three areas. A known example of such schemes is the P2PRep [10] that we discuss later in this section.

Most research papers on the subject do not make a distinction between the reputation of the peer as a network node and the reputation of a data resource such as a file (Damiani et al. [11] is one of the few exceptions). We would like to draw the distinction between these two reputations. The peer reputation is based on behavioral metrics. Such metrics could measure the upload to download ratio or the time that the peer is staying online and transferring queries in the network for example. The peer reputation is the type of reputation that can be used to solve the problem of free-riding.

The data resources reputation focuses on the quality of data resources such as integrity of the file or the content quality. The resource reputation can be subjective to the user's own opinion about the resource. The resource reputation may be, but does not have to be, directly linked to the peer providing it. The resource reputation will then define the desirability of the resource and the peer reputation will define the quality of service the peer may get back from the network on the basis of what the peer is providing.

A. P2PRep

P2PRep [10] is a reputation scheme for Gnutella. The protocol involves four phases. In phase 1, a peer p sends a query for the resource and gets in response $queryHit$ messages which includes a list of the IP addresses of the peers who have the queried file, their IDs, public keys, and numbers to download from and other information such as the download speed in $Kbps$. In phase 2, p chooses from a set of peers T from $querHit$. p then asks its neighbors (neighborhood is defined by Gnutella) for their votes for all the peers in set T. In response, p gets a $PollReply$ message. $PollReply$ includes: the public key PK_i of voter i and their vote signed with the private key of the voter SK_i . The vote contains the IP address, port number, $peer\_ID$ of the voter and the vote itself. In phase 3, p assesses the votes. If the response was received from a new voter, the voter will be contacted to verify that the vote really came from the IP address specified in the message. Finally, in phase 4, p decides which peer it download the file from based on the connection speed, votes received weighted by the credibility of the voter (each peer maintains the local credibility score for its neighbors). Once p has chosen peer q to download from, p needs to verify the link between the peer's ID and the public key received in phase 1. The peer's ID is the digest of the corresponding public key. In order to verify that the peer p has the matching private key, a challenge-response mechanism is used.

B. Eigen Trust

EigenTrust is a system of deriving global trust values from local trust values[12]. The global reputation of peer j is the aggregation of the local reputation scores given to j weighted by the global reputation of the peers who gave these reputation scores to j. Let's say that there are two peers interacting i and j. i decides how to rate the transaction with j. EigenTrust does not define what parameters i would use to rate j. Let s_{ij} be the local trust value given by i to j. s_{ij} can be calculated as the difference between the number of good (i satisfied) and bad (i unsatisfied) transactions.
$s_{ij} = sat(i,j) - unsat(i,j)$
s_ij is then normalised as follows:
$c_{ij} = \frac{max(s_{ij},0)}{(\sum_j max(s_{ij},0)}$
where c_{ij} is the normalized local trust value. If i wants to know the trust value for a peer k, i asks its own neighbors about k for their local trust values for k (i has the local trust values for its neighbors). i then weights the received values by the normalized trust value it has for each of the peers who answered: $t_{ik} = \sum_j c_{ij}c_{jk}$. The algorithm is generalised to cover more peers that are more than one hop from i (for which i's immediate neighbors do not have local trust values).

C. TrustGuard

The TrustGuard reputation scheme uses the oscillation guard algorithm to stabilise the reputation scores over time [13]. The oscillation guard algorithm uses the proportional-integral-derivative (PID) controller commonly used in automatic control systems.

First, raw reputation values $R(i)$ are calculated, where i is a time interval. $H(i)$, the history component, is calculated using the formula:
$H(i) = \sum_{k=1}^{maxH} R(i-k)\frac{w_k}{\sum_{k=1}^{maxH} w_k}$
where $maxH$ is a maximum history interval over which the reputation values are being stored, w_k is chosen optimistically or pessimistically. If chosen optimistically, then w_k would be calculated as $w_k = p^{k-1}$ where $p < 1$. Choosing $p < 1$ gives the higher weighting for the recent feedback. The pessimistic choice of w_k gives the higher weighting to "bad" scores. It is calculated using the formula:
$w_k = \frac{1}{R(i-k)}$
The next step is the calculation of the derivative component $D(i) = R(i) - H(i)$

Authorized licensed use limited to: UDESC - Universidade do Estado de Santa Catarina. Downloaded on September 08,2021 at 17:58:40 UTC from IEEE Xplore. Restrictions apply.

The current aggregated reputation score for the peer is the sum of all the trust values $TV(i)$ over all the time intervals, where $TV(i)$ is calculated using the formula:

$$TV(i) = aR(i) + bH(i) + fD(i). \quad (1)$$

So for example if the reputation score is raising up too fast oscillation guard reacts to this situation by giving higher weighting to the history component (i.e. increase a) and if the score is negative oscillation guards reacts by giving higher weighting to the derivative component (i.e. increase f).

III. Our Architecture: Rabit

The aim of Rabit is to offer a reputation scheme to evaluate the quality of the service provided by each peer. The reputation score is actually an incentive reputation since it can be used to provide a lower quality of service to peers with lower reputation scores. The reputation is based on past votes. Each peer gets a vote from the transaction partner. In order to make our scheme as generic as possible, we do not specify which parameters are included in the votes, but rather focus on how to securely aggregate the votes and securely query the network for the reputation scores. We assume that the vote is a value v assigned to a peer by another after a transaction between them based on QoS metrics. Votes can be positive, negative or 0. The aggregated reputation score can be either a positive value or zero. This choice is justified by the fact that if we allow negative reputation scores. Then, any peer with negative reputation will simply change its identity to become a new peer with the default initial score. The initial score has to be zero. Otherwise, if the initial score is a negative value, new peers would not be trusted by others. On the other hand, we have decided to allow negative votes. Negative votes will be used to "punish" bad peers. We limit how often peers can send votes in order to reduce the traffic.

The reputation expires/decreases after a period of time t. The expiry period can be as short as one or few weeks. The reason for the relatively short expiration period is the fact we need to encourage peers to constantly contribute to the network.

Rabit distinguishes three entities: the requester, the provider, and the score aggregator. This division is purely logical and is based on the role the peer is playing at a particular moment in relation to the processes taking place. In Bittorent, the peer can download and upload at the same time. So at the same time the peer can play the roles of the requester downloading pieces of the file, the provider when it uploads the pieces, and aggregator if the peer is currently aggregating the scores for another peer.

The aggregation algorithm we use is based on the oscillation guard algorithm described in [13]. We use cryptography for integrity protection of the votes and the aggregated scores themselves. Our aim is not just to create a lightweight, accurate and secure system but also to minimise its overhead. This is why we decided to use neither the GossipTrust, EigenTrust, nor P2PRep. Indeed, the GossipTrust [14] network traffic overhead in gossip message spreading is $O(nlog_2 n)$. EigenTrust's complexity is $O(n^2)$ [12]. The overhead in Gnutella for using P2PRep is approximately doubling the traffic [15] in an already busy environment. In addition to the aggregation algorithm, we propose a technique to choose the reputation aggregators. The peer has the choice to decide how trustworthy the aggregated scores are.

The proposed reputation scheme is designed to be fully decentralized. Any peer involved in the reputation mechanisms will need to generate a private and a public key that will be used for signing and identifying the peer. Each peer has unique ID which the peer is known by in the DHT table. We specify that the unique ID is SHA1 hash of the peer's public key. Taking into account the length of the key (160 bit), it is unlikely that two peers would have the same unique ID. The peer has to keep this ID as long as the peer wants to use the reputation associated with that key.

We store the reputation for each peer p on r locations. The peers responsible for these r locations will act as aggregators for p and will store and aggregate its scores. The r locations are chosen as follows:

- Write the provider's ID in binary form.
- Flip the last (most significative) bit of the ID leaving the rest of the ID unchanged. The generated ID will give us the first location at which the reputation will be stored.
- Flip the next to last bit of the *original* ID. The generated ID is the second location.
- Flip the last and next to last bits of the original ID. The generated ID is the third location.
- Apply this process until r locations are generated.

The reason for choosing the keys this way is to maximize the distance between the keys so to guarantee that no aggregator will happen to be responsible for more than one location. Indeed, the distance between peers in the Kademlia is the result of the XOR operation performed on the peers IDs [16]. So choosing the reputation locations as described above maximizes the distance between the aggregators. The aggregators will aggregate votes and store the outputs. Strictly speaking, more than one peer would be responsible for the key due to the replication used in Kademlia. Using the above algorithm for determining the locations, any peer q can find the aggregators locations for p if q knows the ID of p.

With the 160 bit ID, there is a maximum of $2^{160} - 1$ possible locations. The number of locations r has be chosen big enough to ensure the availability of aggregators, and has to be small enough to avoid flooding the network with messages exchanged with aggregators. The number of aggregators can also be changed dynamically depending on the network conditions.

We define five main phases in our scheme: requesting peers, choosing the best peers, transaction, voting and aggregating the votes (cf. Fig. 1). In what follows we describe each phase. The transaction phase is the same as

Authorized licensed use limited to: UDESC - Universidade do Estado de Santa Catarina. Downloaded on September 08,2021 at 17:58:40 UTC from IEEE Xplore. Restrictions apply.

defined by BitTorrent and does not need to be redefined.

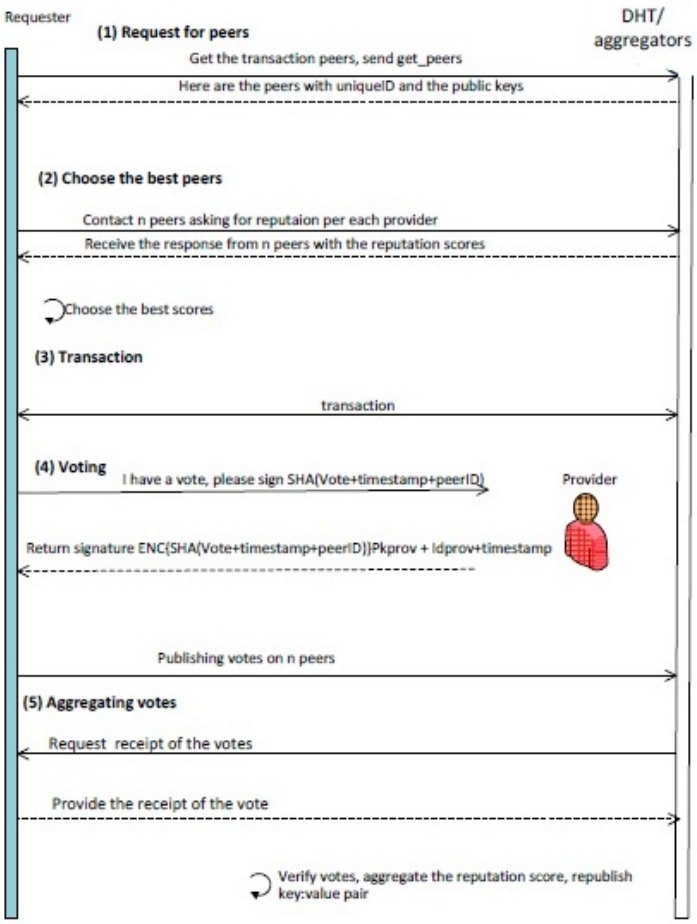

Fig. 1. Protocol flow

A. Requesting peers

The first step is the normal *get_peer* for BitTorrent. A requester, wishing to join the swarm, sends *get_peers* request to the node responsible for the *.torrent* file to get information about the swarm members. The information received includes: ID, IP address and port number for each peer in the swarm. We add the peers public keys to these messages. After the peer set has been received the next step is to retrieve the reputation information from the aggregators. The requester sends $find\_value$ request with the peer's reputation locations determined as described earlier. As shown in [16], the search complexity of the Kademlia algorithm is $O(logn)$. If each peer's score is aggregated by r aggregators, the number of lookups necessary is $rlog_2n$, where n is the number of the peers currently using BitTorrent (not only the peers in the swarm). Since the r is a constant, the complexity of the algorithm is $O(logn)$. At this stage the peer caches contact information of all the peers in the set and their aggregators for the duration of the transaction. This is to avoid the overhead of locating the aggregators again when sending the votes to them after the transaction is done.

B. Choosing the best peers

The requester needs to find out the reputation of the potential providers. Once the provider's r number of reputation scores have been retrieved, the requester calculated the average of all scores, discards the votes further then the standard deviation away considering these scores suspicious and calculates the average of the remaining scores. The peers with the highest reputation scores are chosen and the transactions (download/upload) start.

C. Voting

As a result of the transaction each peer is expected to give a vote about its transaction partner. The requester (downloader) calculates the hash of the vote, combined with a timestamp and the ID of the provider (uploader), and sends the hash to the provider. The requester sends the hash of the vote not the vote itself to stop the provider refusing to sign the vote in case the provider does not like the vote. The provider signs the vote and sends it back to the requester. The requester then sends the vote with the timestamp, provider's ID and the provider's signature to the r aggregators of the provider. The requester also stores the vote receipt for one hour to provide the receipt to any future requester of the same provider if required. After one hour, the vote expires and can be deleted. The contact details of the aggregators have already been cached so no peer lookups necessary at this stage. We assume that the number of aggregators chosen is large enough to guarantee that a high percentage of the aggregators will still receive the votes. The requester can send the vote for the same peer only once in every 15 min. This condition of "saving" the votes reduces the amount of the traffic passed in the network.

D. Votes aggregation

Once the vote is received by the aggregator, the aggregator verifies its signature. The public key can be found directly from the provider. Any vote that fails the verification is discarded. The aggregator also checks that not more than 1 in 15 min messages were sent from the same voter about the same provider as well as verifies the freshness of the timestamp. The aggregator aggregates the votes following the TrustGuard algorithm as described in the previous section. The votes from the last time interval are aggregated by first calculating the mean value of all votes and taking only votes which are one standard deviation away from the mean. This will discard extreme votes considered suspicious. We recalculate the mean after discarding extreme votes. We then use formula 1 proposed in Trust Guard to aggregate the scores.

Each vote is received by the aggregator as a $STORE$ request sent by the requester (downloader) as a pair $(location, vote)$. The aggregators receive the votes and aggregate them every 15 to 30 minutes. If there are no new votes the score is aggregated when the $(location, vote)$ pair is republished until the score expiration time. The keys are republished once an hour [16]. One of the main

Authorized licensed use limited to: UDESC - Universidade do Estado de Santa Catarina. Downloaded on September 08,2021 at 17:58:40 UTC from IEEE Xplore. Restrictions apply.

problems in P2P network is the high churn. So to protect against the peers suddenly leaving the network and taking all the $(location, vote)$ pairs with them, Kademlia uses a data replication mechanism [16]. Azureus, a popular BitTorrent software client, uses 20 peers replication factor [17]. When a new vote is sent to a specified ID it is also replicated to 20 other peers. This replication will also reduces the chance of a malicious peer subverting the reputation.

IV. Discussion

In this section, we discuss how Rabit tackles some of the most known P2P attacks, namely: sybil attack, whitewashing, self-promoting, slandering, collusion and traitors.

A. Sybil Attack

The essence of this attack is the generation of multiple identities with the view of subverting the system using the fake identities. One of the ways to defend against the Sybil attack is the introduction of strong identities. The reputation schemes such as RCert [18] and DMRepDI [15] rely on the peers using certificates. We offer to link the voter's unique ID to the proof of the performed transaction. A similar notion of the need for the transaction proof exchange is mentioned in the TrustGuard framework [19]. In our design we do not reinforce strong identities. Instead we try to decrease the benefit from owning the multiple identities. We use cryptography to link the current reputation score to the uniqueID and the public key of the peer. If a peer changes the public key and therefore the uniqueID the reputation is lost. We also store the voters' IP addresses and verify the votes came from the correct IP as well as generating the votes receipts. The most important defence we implement against Sybil attack is that the aggregators of a peer are picked by flipping bits of peer's ID. If a malicious user wants to be, say, an aggregator for that peer, they need to create a peer ID that is responsible for one of the r locations. But because peer IDs are generated by hashing their public keys, making the output completely random (as long as a good hash function is used), it is impossible for the attackers to have any control on the ID of the peer(s) they own.

B. Whitewashing

Whitewashing is the change of identity to avoid a bad reputation [20]. In our reputation schemes the nodes can issue positive or negative votes but the aggregated reputation score cannot fall below zero. In defence to this decision we offer the following reasons: the peer has no need to change the identity to avoid a bad reputation since there is no bad reputation; the aim of our reputation is to encourage the peers to share physical resources with the network therefore the lowest score corresponds to no resources shared yet. The lowest reputation, zero, is the same as the score for the peer which has just joined the network. We are, however, giving a sharing peer a chance to start building the reputation initially by dynamically adjusting the coefficients in the score oscillation guard formula offered in TrustGuard and slowing the reputation score building as the peer's score grows.

C. Self promoting

Hoffman et al. [9] also identify self promoting attacks where the peer tries to increase their own reputation by falsely increasing the score. We use cryptography to protect the votes. We also aggregate the score on randomly selected peers, making it impossible for peers to fake their reputation scores.

D. Slandering

Slandering is falsely reducing the reputation of honest peers [9]. It is very difficult to stop the peers lying and therefore most reputation schemes assume if the number of voters is large enough, the received score would be relatively accurate. We use two mechanisms to try and protect from lying peers. First we filter the extreme scores considering them unreliable by computing a mean value of all the votes and only considering the mean of the votes one standard deviation away from the mean. Second, we also specify that the voter can only send the vote once in the specified period of time (15 to 30 minutes) and has to obtain the proof of the transaction from the provider. This would stop a malicious peer generating and successfully submitting multiple fraudulent votes and therefore reduce the negative influence of a lying peer on another peer's final reputation scores. We discuss slandering in coalitions in the collusion section.

E. Collusion

Collusion attack is an attack in which a set of peers cooperate in order to carry out some malicious action. It is very difficult to protect against this sort of attacks and is currently considered mostly unsolved [21]. One of the examples is a group of malicious peers publish a .torrent file and joining the swarm for the file. Any new peer joining is trapped. While this sort of attack seems difficult to protect against we can suggest a few countermeasures. First of all, peers in our reputation scheme can see their own reputation values (by simply contacting the aggregators) and can therefore leave the swarm if suspecting foul play (e.g. by seeing their reputation score drop quickly).

The colluding peers can also promote each other's reputation. While it is difficult to eliminate this type of attack, the design of our system slows it down. Indeed, only one vote is accepted from a single peer within a defined period of time.

There is still a question of the peers building the reputation for each other by colluding in the clique. As a future work for our reputation schemes we suggest the aggregators reacting to the groups of the peers exclusively cooperating with each other. One of the ways of achieving the detection is examining the proofs of where the transaction has originated from.

Authorized licensed use limited to: UDESC - Universidade do Estado de Santa Catarina. Downloaded on September 08,2021 at 17:58:40 UTC from IEEE Xplore. Restrictions apply.

F. Traitors

Misbehaving peers are also called traitors [20]. Traitors build the reputation for some period of time then use it later to abuse the system. We adopt the score oscillation algorithm offered by the TrustGuard [19] that decreases the impact on a reputation score from such peers. This algorithm combines three components using three coefficients. The first component represents the current score, the second one represents the historic scores and the last element is a derivative between the current score and the history which would have little effect if the peer is behaving evenly and pulls the total reputation score down when peers start misbehaving.

V. Conclusion

In this paper, we proposed a new scalable and secure reputation framework for BitTorrent. This framework provides secure mechanisms for sharing votes and retrieving reputation scores of any peer participating in the framework. We have analysed the security of this scheme and proven that it is invulnerable to most of the common P2P attacks.

Further works could focus on the study the effect of various parameters of the performance of our scheme. In particular, it is very interesting to study the effect of the number of chosen aggregators (number of keys) on the probability of finding an aggregator online, as a function of time.

References

[1] G. Camarillo, "Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability," RFC 5694 (Informational), Internet Engineering Task Force, Nov. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5694.txt

[2] B. Cohen, "The BitTorrent protocol specification version 11031," http://www.bittorrent.org/beps/bep_0003.html, 2008, [Online; accessed 15-July-2011].

[3] A. Loewenstern, "DHT protocol," http://www.bittorrent.org/beps/bep_0005.html, 2008, [Online; accessed 15-July-2011].

[4] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decis. Support Syst.*, vol. 43, no. 2, pp. 618–644, Mar. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.dss.2005.05.019

[5] M. Karakaya, I. Korpeoglu, and O. Ulusoy, "Free riding in peer-to-peer networks," *IEEE Internet Computing*, vol. 13, pp. 92–98, 2009.

[6] D. Hughes, G. Coulson, and J. Walkerdine, "Free riding on gnutella revisited: the bell tolls?" *In IEEE Distributed Systems Online*, 2005.

[7] B. Cohen, "Incentives build robustness in BitTorrent," http://www.bittorrent.org/bittorrentecon.pdf, 2003, [Online; accessed 15-March-2012].

[8] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer, "Free riding in bittorrent is cheap," in *In HotNets*, 2006.

[9] K. Hoffman, D. Zage, and C. Nita-Rotaru, "A survey of attack and defense techniques for reputation systems," *ACM Comput. Surv.*, vol. 42, no. 1, pp. 1:1–1:31, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1592451.1592452

[10] E. Damiani, S. De Capitani Di Vimercati, S. Paraboschi, and P. Samarati, "Managing and sharing servants' reputations in p2p systems," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 15, no. 4, pp. 840 – 854, july-aug. 2003.

[11] E. Damiani, D. C. D. Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *In Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM Press, 2002, pp. 207–216.

[12] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the 12th international conference on World Wide Web*, ser. WWW '03. New York, NY, USA: ACM, 2003, pp. 640–651. [Online]. Available: http://doi.acm.org/10.1145/775152.775242

[13] M. Srivatsa, L. Xiong, and L. Liu, "Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks," in *WWW*, 2005, pp. 422–431.

[14] R. Zhou, K. Hwang, and M. Cai, "Gossiptrust for fast reputation aggregation in peer-to-peer networks," *IEEE Trans. on Knowl. and Data Eng.*, vol. 20, no. 9, pp. 1282–1295, Sep. 2008. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2008.48

[15] P. Dewan and P. Dasgupta, "P2p reputation management using distributed identities and decentralized recommendation chains," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1000–1013, 2010.

[16] P. Maymounkov and D. Mazières, "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*, ser. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65. [Online]. Available: http://dl.acm.org/citation.cfm?id=646334.687801

[17] S. A. Crosby and D. S. Wallach, "An analysis of bittorrents two kademlia-based dhts," 2007.

[18] B. C. Ooi, C. Y. Liau, and K.-L. Tau, "Managing trust in peer-to-peer systems using reputation-based techniques," in *In Proc. of WAIM*, 2003, pp. 2–12.

[19] M. Srivatsa, L. Xiong, and L. Liu, "Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks," in *Proceedings of the 14th international conference on World Wide Web*, ser. WWW '05. New York, NY, USA: ACM, 2005, pp. 422–431. [Online]. Available: http://doi.acm.org/10.1145/1060745.1060808

[20] S. Marti and H. Garcia-Molina, "Taxonomy of trust: Categorizing p2p reputation systems," *Computer Networks*, vol. 50, no. 4, pp. 472–484, 2006.

[21] G. Swamynathan, K. Almeroth, and B. Zhao, "The design of a reliable reputation system," *Electronic Commerce Research*, vol. 10, pp. 239–270, 2010, 10.1007/s10660-010-9064-y. [Online]. Available: http://dx.doi.org/10.1007/s10660-010-9064-y

Authorized licensed use limited to: UDESC - Universidade do Estado de Santa Catarina. Downloaded on September 08,2021 at 17:58:40 UTC from IEEE Xplore. Restrictions apply.