

Securing Peer-to-Peer Content Sharing Service from Poisoning Attacks

Ruichuan Chen^{1,3}, Eng Keong Lua², Jon Crowcroft², Wenjia Guo^{1,3}, Liyong Tang^{1,3}, Zhong Chen^{1,3}

¹Institute of Software, School of EECS, Peking University, China

Email: {chenrc, guowj, tly, chen}@infosec.pku.edu.cn

²Computer Laboratory, University of Cambridge, United Kingdom

Email: {eng.keong-lua, jon.crowcroft}@cl.cam.ac.uk

³Key Laboratory of High Confidence Software Technologies, Ministry of Education, China

Abstract

Poisoning attacks in the Peer-to-Peer (P2P) content sharing service have become a serious security problem on the global Internet due to the features of P2P systems such as self-organization, self-maintenance, etc. In this paper, we propose a novel poisoning-resistant security framework based on the notion that the content providers would be the only trusted sources to verify the integrity of the requested content. To provide the mechanisms of availability and scalability, a content provider publishes the information of his shared contents to a group of content maintainers self-organized in a security overlay, so that a content requestor can verify the integrity of the requested content from the associated content maintainers. Two defense functions are first carried out — filtering out malicious activities and selecting the authentic content version. Then, the content requestor can perform the content integrity verification while downloading and take prompt protection actions to handle content poisoning attacks. To further enhance the system performance, we devise a scalable probabilistic verification scheme. The evaluation results illustrate that our framework can effectively and efficiently defend against content poisoning in various scenarios.

1 Introduction

Peer-to-Peer (P2P) content sharing service has grown in significance on the Internet, both in terms of the number of participating users and the traffic volume [14]. However, due to the self-organization and self-maintenance nature of P2P overlay networks, each participating user has to manage the potential risks involved in the application transactions without adequate experience and knowledge about other users. Many studies indicated that P2P content sharing systems are highly vulnerable to content poisoning. Specifically in KaZaA [12], even more than 50% of

the copies of many recent popular songs are poisoned [16].

In a typical content poisoning attack, the poisoners first corrupt the target content with the intention of degrading its quality or rendering it unusable, and then inject a massive quantity of poisoned contents into the P2P content sharing system. Unable to distinguish authentic content from poisoned content, unsuspecting P2P users download the poisoned content into their own shared folders, from which the other P2P users may download later without knowing that the content has been poisoned. As a result, the poisoned content spreads through the P2P content sharing system at extraordinary speed, i.e., the poisoning attack affects the quality of service and/or causes the content sharing service to be unusable.

Generally, the simple digital signature scheme can be used to defend against content poisoning in many Internet applications. However, due to the lack of centralized trusted content distribution authorities in the P2P content sharing systems, the applicability of the signature scheme is questionable. In this paper, we develop a novel poisoning-resistant security framework for P2P content sharing service based on the fundamental fact that the content providers are the only sources to accurately distinguish poisoned contents and verify the integrity of the requested contents. Our proposed framework is built upon a distributed hash table (DHT) based overlay to provide the mechanisms of availability and scalability. In our framework, a content provider disseminates his shared contents' information to a group of content maintainers self-organized in the DHT-based overlay. This allows a content requestor to perform integrity verification on the requested content by first looking up in the DHT-based overlay for the associated content maintainers who have maintained the corresponding content information. Then, the content requestor can execute the filtering process to filter out malicious activities using a threshold scheme, followed by a selection strategy to choose the authentic content version for downloading. Our security framework allows content requestors to verify the in-

egrity of the requested content while downloading and take prompt protection actions to handle content poisoning attacks. To reduce the verification overhead, we devise a scalable probabilistic verification scheme in which each content requestor has the capability of flexibly adjusting the false positive rate of integrity verification according to the trade-off between integrity assurance and verification overhead.

We conduct simulation studies on several synthetic P2P overlay networks with different network, user, content and execution models. The evaluation results show that our poisoning-resistant security framework can effectively and efficiently defend against content poisoning in various different scenarios. In addition, our security framework can easily be extended to incorporate with existing defenses against man-in-the-middle (MITM), Sybil [7] and Denial-of-Service (DoS) attacks.

Outline. The rest of this paper is organized as follows. We give an overview of related work in section 2. The details of our poisoning-resistant security framework are described in section 3. Section 4 presents the simulation methodology and evaluation results. Finally, we conclude with a discussion of incorporating our framework with existing defenses against MITM, Sybil and DoS attacks in section 5.

2 Related work

Currently, many reputation models have been proposed to address the problem of content poisoning in P2P content sharing systems. In general, these reputation models can be grouped into three categories: *peer-based* models, *object-based* models and *hybrid* models. In peer-based reputation models, e.g., EigenTrust [11], PeerTrust [28] and Scrubber [4], genuine users collectively identify content poisoners by computing a reputation score for each user, and then isolate these poisoners from the system. However, the studies in [8, 27] implied that these peer-based models are insufficient to defend against the poisoning attack. In object-based reputation models, e.g., Credence [27], genuine users determine the object authenticity through secure tabulation and management of endorsements from other users. Aiming at combining the benefits of both peer-based and object-based models, several hybrid reputation models, e.g., XRep [6], X²Rep [5] and extended Scrubber [3], have been further presented. Nevertheless, due to the fact that most of the participating users in P2P content sharing systems are rational in seeking to maximize their individual utilities, the reputation models are greatly penalized by the lack of reliable user cooperation.

Besides the above reputation models, micropayment techniques, e.g., MojoNation [20], can also be utilized to counter the poisoning attack by imposing a cost on content poisoners — to inject poisoned content into the system they

should first commit a certain amount of resources. Furthermore, Habib *et al.* in [9] developed an integrated security framework to verify content integrity in P2P media streaming. However, this framework requires centralized supplying peers. Recently, Michalakis *et al.* in [19] presented a “Repeat and Compare” system to ensure content integrity for P2P content distribution networks by detecting poisoned contents through attestation records and sampled repeated execution.

3 Poisoning-resistant security framework

To elaborate the following design rationale clearly, we first introduce the P2P terminology. Then, we describe the novel poisoning-resistant security framework for P2P content sharing systems.

3.1 P2P terminology

In a real-world P2P content sharing system, each *content* is associated with a specific *file*. A file can generally have multiple different *versions*, each of which is published by a group of providers. Specifically, each version has an *identifier*, which is typically a hash value of the corresponding file data (and metadata); therefore, additional versions are produced if any bits of the file data are modified.

3.2 Design rationale

To provide the mechanisms of availability and scalability, the underlying structure of our poisoning-resistant security framework for P2P content sharing systems is a DHT-based overlay. We utilize Chord [26] as the dedicated underlying DHT-based overlay. We can also conveniently utilize another DHT-based overlay (e.g., CAN [22], Tapestry [31], Pastry [24] or Kademia [18]) as an alternative.

3.2.1 Publish

Since the underlying structure is the Chord overlay, we utilize SHA1 to assign each user and file an identifier. The *user identifier* of a participant is chosen by hashing the participant’s IP address (or the other unique and constant identity), while a *file identifier* is produced by hashing the filename. These identifiers are ordered in an identifier circle. The *information* of file F is assigned to the first user whose identifier is equal to or follows the F ’s file identifier in the identifier circle. This user is called the successor of file F , denoted by $successor(F)$. Such structure tends to balance the load on the system, since each user maintains the information of roughly the same number of files, and there should be little information movement under churn.

Table 1. The information of a specific file

	Version Identifier	Digest List	Provider Identifier List
1	VI_1	DL_1	PIL_1
2	VI_2	DL_2	PIL_2
...
i	VI_i	DL_i	PIL_i
...
n	VI_n	DL_n	PIL_n

In our security framework, a *provider* P , who wants to publish a specific file F , first divides the file F into b data blocks and, then, he computes the digest of each block to construct a *digest list*, i.e., $\{digest_i\}_{i=1}^b$. Here we could further utilize the homomorphic hashing technique [13] to allow future file requestors to perform order-independent verification on these data blocks. Afterwards, according to F 's file identifier, the provider P maps the information of file F (including the file identifier, digest list and his own user identifier) onto the associated *maintainer* M , i.e., *successor*(F). Finally, once the maintainer M has received such file information, he can additionally take hash of the entire digest list as a *version identifier* of file F . In particular, since several providers may individually publish a number of versions of file F , the information of F stored at the maintainer M is shown in Table 1.

However, a maintainer may be offline and a malicious or compromised maintainer has the capacity of tampering with the file information maintained by himself. Therefore, we map the information of a specific file F onto m maintainers $\{M_i\}_{i=1}^m$ to provide the mechanisms of availability and scalability, i.e., each file F corresponds to a unique group of maintainers.

$$M_i = \text{successor}(\text{filename}|i), \quad 1 \leq i \leq m \quad (1)$$

where

- M_i : The i th maintainer of the file F .
- *filename*: The name of the file F .
- m : The number of maintainers associated with the file F .
- “|” : The concatenation operator.

Firstly, due to the essential feature of DHT-based overlay, these m maintainers are distributed in the identifier circle uniformly at random, so most of them are genuine with high probability in a normal network environment; therefore, we can design a threshold scheme to filter out the malicious activities (we will describe this later). Secondly, since

each maintainer only knows the maintained file identifier (i.e., hashed filename) but not the filename, according to the expression 1 even a malicious maintainer cannot gain the user identifiers of other maintainers associated with the same file, so he is not able to compromise other maintainers.

3.2.2 Lookup and downloading

In the following, we will describe in detail the lookup and downloading process of our security framework.

Step 1 — Query: A *requestor* U , who wants to acquire a specific file F , issues a query (i.e., F 's file identifier) towards the m associated maintainers in the system according to the expression 1. Each of these maintainers should maintain the information of at least one version of the requested file F . Note that, in this framework, we assume that we have the perfect overlay routing and maintainer discovery.

Step 2 — Response: Subsequently, these m maintainers respond with the corresponding information of the requested file F (see Table 1).

Step 3 — Filtering: On harvesting these responses, the requestor U mixes them and generates a list L consisting of $\langle VI, PI \rangle$ pairs (do not remove duplicate pairs), where the VI and PI denote the version and provider identifiers associated with the requested file F , respectively.

We design a *threshold scheme* here to filter out the malicious activities. Firstly, without loss of generality, we assume that the percentage of malicious and compromised users should not exceed θ_1 ($0 \leq \theta_1 < 0.5$) in the whole system. This implies that though malicious and compromised maintainers may collectively create/delete/modify their maintained $\langle VI, PI \rangle$ pairs, each $\langle VI, PI \rangle$ pair existing in the generated list L for at least $|m \times (1 - \theta_1)|$ times is authentic with high probability. Therefore, we filter the list L through removing the pairs with less than $|m \times (1 - \theta_1)|$ times. Secondly, a genuine provider generally publishes one or only a few¹ versions of a specific file. Hence, if a provider P_i publishes more than θ_2 ($\theta_2 \geq 1$) versions of the requested file F , we should further filter the list L through removing all these $\langle VI, PI \rangle$ pairs associated with the provider P_i . Finally, the requestor U reconstructs the information of the requested file F (see Table 1) by excluding the above two kinds of filtered information.

Step 4 — Selection: After the information filtering, the requestor should select a version V based on his own selection strategy.

Since the size of the associated *provider identifier list* generally reflects the availability of a specific version, in most cases, the requestor U selects the version with the largest provider identifier list. However, the analytical model presented in [8] characterized the impact of various selection strategies, and implied that the “largest” strategy

¹These versions may exist in different shared folders of the provider.

is vulnerable to the poisoning attack as well as the DoS attack. As an alternative, in our framework, the requestor U 's choice is biased towards versions with more providers. That is, the probability of selecting a version is proportional to the size of the associated provider identifier list.

$$\Pr(V_i) = \frac{|PIL_i|}{\sum_{j=1}^n |PIL_j|}, \quad 1 \leq i \leq n \quad (2)$$

where

- V_i : The i th version of the requested file, i.e., the version with the identifier VI_i .
- $\Pr(V_i)$: The probability of selecting the V_i .
- $|PIL_i|$: The size of the provider identifier list of V_i .
- n : The number of versions associated with the requested file.

Step 5 — Downloading and verification: The requestor U starts downloading the data blocks of the selected version V from these associated providers in parallel. Specifically, each provider returns not only the actual data blocks but also the complementary block digests. Once the requestor U receives a data block with the digest D_p , he first locally computes the digest D_u by hashing the received data block and, then, compares the D_p with D_u . If $D_p \neq D_u$, the requestor U simply drops the received data block; otherwise, he additionally matches the D_p against the digest existing in the corresponding digest list of F 's file information (see Table 1). If they are matched, the received data block is accepted, otherwise the data block is dropped. These operations make our framework have the capacity of verifying the integrity of the selected version V while downloading.

In common as described above, aiming at verifying the data integrity, the requestor U verifies all the received data blocks while downloading. To reduce the verification overhead, we propose a **probabilistic verification scheme** which verifies randomly selected blocks instead of all blocks.

In this scheme, the requestor U assumes that a poisoner should tamper with at least r blocks of all b blocks and, moreover, less than r poisoned blocks could be successfully recovered by the error-correcting code (ECC) technique. A thorough investigation of ECC is important, but it is beyond the scope of this paper. Based on this assumption, the requestor U would need to determine the *expected false positive rate (EFPR)* of the probabilistic verification according to the tradeoff between integrity assurance and verification overhead. That is,

$$FPR = \begin{cases} \frac{C_{b-r}^v}{C_b^v} = \frac{(b-r)! \times (b-v)!}{(b-r-v)! \times b!} & \text{if } r + v \leq b \\ 0 & \text{if } r + v > b \end{cases} \quad (3)$$

$$\leq EFPR$$

where

- FPR : The actual false positive rate.
- $EFPR$: The expected false positive rate.
- b : The total number of data blocks.
- v : The number of verified data blocks.
- r : The lower bound of the number of poisoned data blocks.

Now, the requestor U can randomly verify the minimal number of data blocks, denoted by v_{min} , that satisfies the above expression 3.

Step 6 — Build: After the downloading and verification, the requestor U can build the complete requested file from these data blocks. If the file is authentic, the entire transaction ends successfully; otherwise, if the file is unfortunately found to be poisoned, the requestor U should delete it, *blacklist* the selected version V , and then repeat from step 4 until he receives the authentic version or the authentic version is found to be non-existent in the system. In particular, to enhance the effectiveness and efficiency of blacklisting, each participating user may share his local blacklist with other users, thus resulting in a global blacklisting technique which is beyond the scope of this paper.

4 Evaluation

In this section, we first describe the simulation setup, and then we present the key performance metric. Finally, we assess the performance of our poisoning-resistant security framework as compared to the Credence [27] reputation model.

4.1 Simulation setup

To evaluate the performance, we developed a prototype system implementing all our proposed security mechanisms elaborated in section 3. Furthermore, we need to generate several networks with different network parameters — all should follow certain distributions.

Network model: In the following simulations, we choose Chord as the underlying DHT-based overlay of our security framework, and utilize the Overlay Weaver toolkit [25] to construct the simulation testbed. We assume the perfect overlay routing and maintainer discovery. Moreover, the transfer time is assumed to be negligible.

User model: The network is composed of 2,000 users including genuine users and poisoners. A genuine user shares good versions at the start, and then participates in

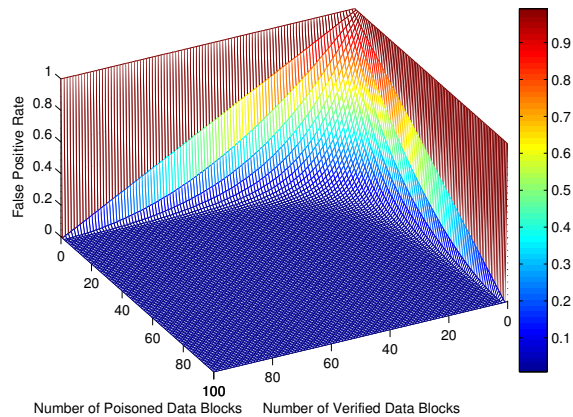


Figure 1. Performance of probabilistic verification.

the overlay network to download the versions of his requested files and share these downloaded versions. In particular, each genuine user is able to independently determine whether to delete the downloaded poisoned versions. In our simulations, we utilize the *incentive for cleaning*, i.e., the probability that a genuine user immediately deletes the downloaded poisoned version, to characterize such determination. Besides, a poisoner shares poisoned versions and participates in the overlay network to spread them — this attempts to undermine the system performance.

Content model: The previous study in [16] reported the existence of a large number of poisoned versions for a single file in the network. In our simulations, there are 100 unique files, each of which has 500 different versions. The information of a file is stored at 10 unique maintainers, and each version is associated with a number of providers which vary over time. At the start, each genuine user shares 50 versions and each poisoner shares 200 versions. Moreover, the versions shared by a participating user are determined by first selecting a certain file and then its version. Specifically, both selections follow Zipf distribution with the parameter $\alpha = 0.8$ [16].

Execution model: Different queries are initiated at uniformly distributed users in the overlay network. In our threshold scheme, the parameter θ_1 is determined by the fraction of poisoners and the parameter θ_2 is set to 5. Specifically, an experimental simulation is composed of 20 simulation cycles, and each simulation cycle is divided into 2,000 lookup cycles. In each lookup cycle, the selection of a specific version to download is done by first selecting a file according to Zipf distribution with the parameter $\alpha = 0.8$, and then choosing a version based on our proposed framework. After each simulation cycle, the number of poisoned downloads is computed. Each experimental simulation is

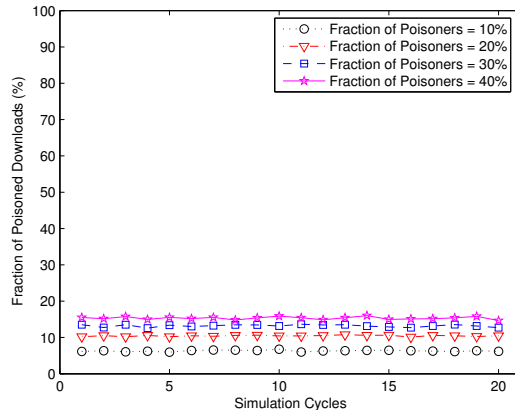


Figure 2. Impact of poisoners, where the false positive rate is 10% and the incentive for cleaning is 100%.

run 5 times and the results of all runs are averaged.

4.2 Performance metric

A well-designed poisoning-resistant security framework should seek to optimize its effectiveness and efficiency in various different scenarios. In the following experimental simulations, we characterize the system performance using the *fraction of poisoned downloads*. It is defined as the fraction of downloads that the requestors acquire poisoned versions from the associated providers during one simulation cycle. Specifically, this metric is computed at the end of each simulation cycle.

4.3 Evaluation results

Probabilistic verification: Before the evaluation of our poisoning-resistant security framework, we first evaluate the effectiveness and efficiency of the probabilistic verification scheme.

We assume that the requested file consist of 100 data blocks. With different numbers of verified data blocks (v) and different numbers of poisoned data blocks (r), we compute the actual false positive rate of the probabilistic verification. Our result shown in Figure 1 demonstrates that the file requestor generally needs to merely verify 30% of all blocks to achieve a low false positive rate. For some other total numbers of data blocks (b), we obtain the similar evaluation results. That is, we can say that the probabilistic verification scheme has the ability of incurring significantly lower verification overhead to provide a high integrity assurance.

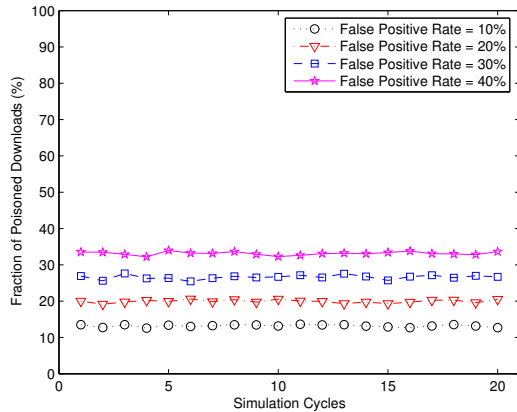


Figure 3. Impact of false positive rate, where the fraction of poisoners is 30% and the incentive for cleaning is 100%.

Impact of poisoners: While fixing the false positive rate to 10% and the incentive for cleaning to 100%, we vary the number of poisoners so that these poisoners make up between 10% and 40% of all the participating users in the system. For each fraction in steps of 10%, we investigate the influence incurred by poisoners. The result shown in Figure 2 clearly indicates a low fraction of poisoned downloads. That is, only a minority of all the downloads will end up with downloading a poisoned version of the requested file. Furthermore, our security framework can work well even in a highly malicious environment with 40% of all users being poisoners.

Impact of false positive rate: In our security framework, each requestor is able to flexibly adjust the false positive rate of probabilistic verification according to the trade-off between the integrity assurance and verification overhead. Generally, the lower false positive rate a requestor expects, the more data blocks he has to verify; whereas the pattern is reversed. In this simulation, we simulate our security framework with the fraction of poisoners and incentive for cleaning being set to 30% and 100%, respectively. Figure 3 shows that our framework can effectively defend against content poisoning in a highly malicious environment. Moreover, a requestor can achieve an acceptable performance even by merely verifying a very small number of data blocks (e.g., false positive rate = 30%). This reflects the efficiency of our framework.

Impact of incentive for cleaning: Incentive for cleaning corresponds to the probability that a genuine requestor immediately removes the downloaded poisoned version — this can be considered as the cleaning of poisoned data in the P2P content sharing system. In this simulation, we vary the incentive for cleaning to further evaluate the ef-

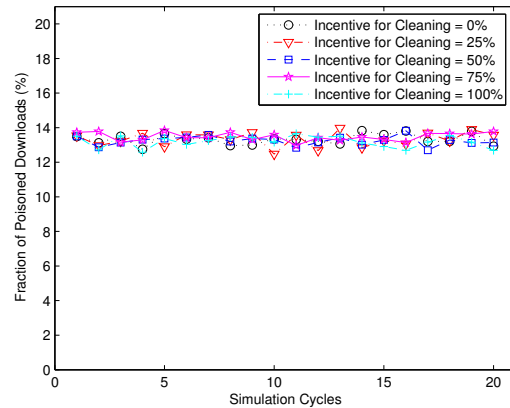


Figure 4. Impact of incentive for cleaning, where the fraction of poisoners is 30% and the false positive rate is 10%.

fectiveness of our security framework. Specifically, with 30% of all users being initialized as the poisoners and the false positive rate being set to 10%, we compute the fraction of poisoned downloads. Interestingly, the result shown in Figure 4 demonstrates that different incentives for cleaning do not significantly influence the system performance. Our analysis takes two factors into consideration. First, a requestor activates his own cleaning module only after he has already downloaded a poisoned version. Second, our security framework is able to filter out most of the poisoned versions before/while downloading and, therefore, a requestor downloads poisoned versions with low probability. Hence, no matter which cleaning strategy is adopted by these requestors, it is difficult for the poisoned versions to spread throughout the content sharing system.

Efficiency evaluation: The goal of poisoners is to trick requestors into repeatedly downloading poisoned versions of the requested file. The previous three experimental simulations have validated that our security framework can effectively defend against content poisoning. Moreover, they also indicate our framework’s efficiency. As shown from Figure 2 to Figure 4, the evaluation results definitely show that, with our security framework, only a small number of downloads end up with downloading poisoned versions even from the start of these simulations. That is, the convergence time of our framework is very short, and it is able to efficiently defend against content poisoning in various scenarios.

Comparison: Recent poisoning defenses [9, 19] are generally deployed in content distribution systems, and most of the poisoning defenses deployed in content sharing systems are based on reputation models. Among these reputation models, Credence [27] is an ingenious model de-

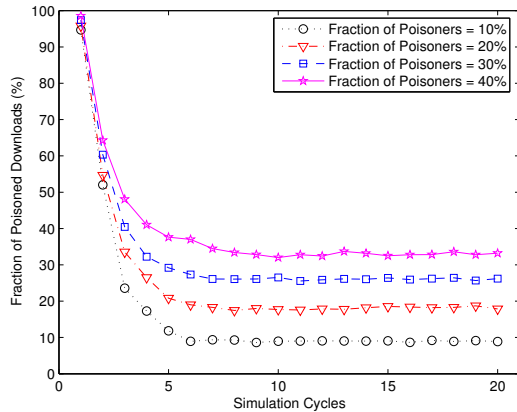


Figure 5. Performance of Credence, where the incentive for cleaning is 100%.

ployed on a live network and, moreover, its scenario is much more similar to ours. Therefore, we compare the performance of our security framework with that of Credence.

In Credence, the participating users assign reputation ratings to the downloaded versions based on their own judgement of the versions’ authenticity. Specifically in our simulation, a genuine user always gives an accurate rating to each version he has ever downloaded; however, a poisoner tries to undermine the system performance and gives the opposite ratings, i.e., he gives positive and negative ratings to poisoned and authentic downloaded versions, respectively. That is, these poisoners value poisoned versions instead of authentic versions to subvert the system. The Credence users utilize a distributed protocol to collect these ratings in the system, and give more weight to ratings from like-minded users according to the statistical correlation. Therefore, Credence users can apply weighted averaging to compute an estimate of a version’s overall reputation.

While fixing the incentive for cleaning to 100%, we simulate the Credence reputation model and evaluate its performance with different fractions of poisoners. Figure 5 indicates that the convergence time of Credence is about 6 to 8 simulation cycles. This is because that during the first several simulation cycles the Credence users could not obtain sufficient ratings from other users to compute an accurate estimate of a version’s overall reputation. When the fraction of poisoners is below 30%, the Credence model can defend against the content poisoning relatively effectively; otherwise its applicability is questionable.

Now, we can compare the performance of our security framework (see Figure 2) with that of Credence (see Figure 5). Our framework outperforms Credence both in terms of effectiveness and efficiency — our framework can effectively defend against the content poisoning from the startup

in various scenarios with different fractions of poisoners. Interestingly, we can further compare the evaluation results of Figure 3 and Figure 5 under the condition of the fraction of poisoners being 30%. Our security framework outperforms Credence when the false positive rate of probabilistic verification is below 30%; otherwise the pattern is reversed. In our framework, each participating user can individually adjust the false positive rate according to his own trade-off between integrity assurance and verification overhead. Based on the result shown in Figure 1, we can safely claim that the false positive rate being 30% or lower is enough for users to verify only a very small number of data blocks to guarantee the content integrity.

5 Conclusion and discussion

Our poisoning-resistant security framework for P2P content sharing systems is able to defend against the content poisoning attack effectively and efficiently. However, some other types of attacks can also be mounted against our security framework in real-world P2P content sharing systems, such as MITM attack, Sybil attack [7] and DoS attack.

Generally, an MITM attacker could read, insert and modify arbitrary messages between two genuine users without letting any of the users have the knowledge of the compromised link between them. Thus, the responses generated by the associated maintainers may be unauthenticated to the requestor U . To counter such attack, the requestor U should dynamically maintain a trusted group to perform multiple independent exchanges originating from different trusted group members (similar to the mechanism described in [2]); then, the requestor U can execute the simple majority voting or Byzantine agreement protocol [15] to obtain the actual responses.

Another important security vulnerability is the Sybil attack — an attacker associates itself illegitimately with an arbitrary number of identities by impersonating other users or claiming false identities. Under Sybil attacks, our proposed filtering and selection mechanisms will be invalid. To protect against Sybil attacks, we could utilize the social network among user identities [29, 30], or make use of the unified security scheme through cross-pollination of identity-based cryptography and online secret-sharing methods [17]. As an alternative, we could also adopt the computational puzzle scheme to defend against the Sybil attack [1, 23].

In general, on the Internet, a DoS attack is an attack in which one or more users attempt to thwart genuine users from having access to legitimate services [10]. Specifically in our security framework, the providers and maintainers of popular contents may be flooded with huge amounts of requests, i.e., our security framework may suffer from the DoS attack. Currently, several countermeasures have been proposed. Among them, the computational puzzle scheme

is also an ingenious solution, and it is particularly well-suited for defending against the DoS attack [21]. Here each user has to compute moderately expensive but not intractable puzzles in order to gain access to the resources allocated by providers and maintainers.

Bearing these in mind, we plan to incorporate our poisoning-resistant security framework with the above security schemes to further improve its effectiveness, and then extend its application field to some other kinds of network services, e.g., distributed storage service.

6 Acknowledgment

We would like to thank our shepherd, Aleksandra Kovacevic, as well as the anonymous reviewers for their valuable comments. This work was supported in part by the National Natural Science Foundation of China under grant No. 60773163.

References

- [1] N. Borisov. Computational puzzles as sybil defenses. In *Peer-to-Peer Computing*, pages 171–176, 2006.
- [2] R. Chen, W. Guo, L. Tang, J. Hu, and Z. Chen. Scalable byzantine fault tolerant public key authentication for peer-to-peer networks. In *Euro-Par*, 2008.
- [3] C. P. Costa and J. M. Almeida. Reputation systems for fighting pollution in peer-to-peer file sharing systems. In *Peer-to-Peer Computing*, pages 53–60, 2007.
- [4] C. P. Costa, V. Soares, J. M. Almeida, and V. Almeida. Fighting pollution dissemination in peer-to-peer networks. In *SAC*, pages 1586–1590, 2007.
- [5] N. Curtis, R. Safavi-Naini, and W. Susilo. X²rep: Enhanced trust semantics for the xrep protocol. In *ACNS*, pages 205–219, 2004.
- [6] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *ACM Conference on Computer and Communications Security*, pages 207–216, 2002.
- [7] J. R. Douceur. The sybil attack. In *IPTPS*, pages 251–260, 2002.
- [8] D. Dumitriu, E. W. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. In *SIGMETRICS*, pages 38–49, 2005.
- [9] A. Habib, D. Xu, M. Atallah, B. Bhargava, and J. Chuang. Verifying data integrity in peer-to-peer media streaming. In *MMCN*, 2005.
- [10] M. Handley and E. Rescorla. Internet denial-of-service considerations (rfc4732). <http://www.ietf.org/rfc/rfc4732.txt>, 2006.
- [11] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW*, pages 640–651, 2003.
- [12] KaZaA. <http://www.kazaa.com/>.
- [13] M. N. Krohn, M. J. Freedman, and D. Mazières. On-the-fly verification of rateless erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy*, pages 226–240, 2004.
- [14] R. Kumar, D. D. Yao, A. Bagchi, K. W. Ross, and D. Rubenstein. Fluid modeling of pollution proliferation in p2p networks. In *SIGMETRICS/Performance*, pages 335–346, 2006.
- [15] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
- [16] J. Liang, R. Kumar, Y. Xi, and K. W. Ross. Pollution in p2p file sharing systems. In *INFOCOM*, pages 1174–1185, 2005.
- [17] E. K. Lua. Securing peer-to-peer overlay networks from sybil attack. In *ISCIT*, 2007.
- [18] P. Maymounkov and D. Mazières. Kademia: A peer-to-peer information system based on the xor metric. In *IPTPS*, pages 53–65, 2002.
- [19] N. Michalakis, R. Soulé, and R. Grimm. Ensuring content integrity for untrusted peer-to-peer content distribution networks. In *NSDI*, 2007.
- [20] MojoNation. <http://www.mojonation.net/>.
- [21] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. M. Maggs, and Y.-C. Hu. Portcullis: protecting connection setup from denial-of-capability attacks. In *SIGCOMM*, pages 289–300, 2007.
- [22] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.
- [23] H. Rowaihy, W. Enck, P. McDaniel, and T. L. Porta. Limiting sybil attacks in structured p2p networks. In *INFOCOM*, pages 2596–2600, 2007.
- [24] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.
- [25] K. Shudo, Y. Tanaka, and S. Sekiguchi. Overlay weaver: An overlay construction toolkit. *Computer Communications*, 31(2):402–412, 2008.
- [26] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.
- [27] K. Walsh and E. G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *NSDI*, 2006.
- [28] L. Xiong and L. Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans. Knowl. Data Eng.*, 16(7):843–857, 2004.
- [29] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *IEEE Symposium on Security and Privacy*, pages 3–17, 2008.
- [30] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM*, pages 267–278, 2006.
- [31] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, 2004.