Making Contribution-Aware P2P Systems Robust to Collusion Attacks Using Bandwidth Puzzles

Michael K. Reiter[1] **Vyas Sekar**[2]
Zhenghao Zhang[3]

Sep 23, 2008
CMU-CS-08-156

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Department of Computer Science, University of North Carolina, Chapel Hill, NC, USA; reiter@cs.unc.edu
[2]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA; vyass@cs.cmu.edu
[3]Computer Science Department, Florida State University, Tallahassee, FL, USA; zzhang@cs.fsu.edu

We thank Hui Zhang for initial feedback and discussions on this project. This work was supported in part by National Science Foundation grant number 0756998. Zhenghao Zhang was supported in part by the Planning Grant from Florida State University (Project #022684). The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, Carnegie Mellon University, Florida State University, or the U.S. Government or any of its agencies.

Keywords: Peer-to-Peer, Collusion, Ballot Stuffing, Reputation, Incentives, Security

Abstract

Many peer-to-peer (P2P) content-distribution systems reward a peer based on its contribution to the system, specifically the amount of data that this peer serves to others. However, validating that a peer did so is, to our knowledge, an open problem; e.g., in a simple form of "ballot stuffing" attack, a group of colluding attackers can earn rewards by claiming to have served content to one another, when they have not. We propose a simple puzzle mechanism to make contribution-aware P2P content distribution systems robust to such collusion. Our construction is novel in that it both ties solving the puzzle to possession of content and, by issuing puzzle challenges simultaneously to all parties claiming to have the same content, prevents one content-holder from solving many others' puzzles. We provide two bounds (in the random oracle model) for adversaries' ability to defeat our puzzle scheme, one closed-form bound and one more precise, efficiently computable, but non-closed-form bound. We additionally evaluate our design in the context of the Maze P2P file-sharing architecture.

1 Introduction

Content distribution via peer-to-peer overlays is becoming increasingly popular; it has even been reported that peer-to-peer file-sharing contributes more traffic volume to the Internet than any other application [14]. Many such systems measure peer contribution and incentivize participation by either providing the peers who contribute more with better performance (e.g., by giving them higher priority in the distribution overlay [41, 36] or providing priority service through server-assisted downloads (e.g., [27, 40]), or through out-of-band mechanisms (e.g., discount coupons, frequent-flier-like rewards [40]). We refer to such P2P systems as *contribution-aware* P2P systems in that they actively measure the contribution of peers and reward them accordingly.

Unfortunately, to our knowledge, all known mechanisms for demonstrating how much data a peer has served are vulnerable to a simple form of "ballot stuffing" [16, 10]. A group of colluding attackers can report receiving service from each other without actually transferring content among themselves. In some systems, this enables these peers to attack the system, e.g., by gaining a powerful position in the distribution overlay and then launching a denial-of-service attack [41, 36]. In others, this enables these peers to get preferential service, or to get high-quality service while contributing only a limited amount of upload bandwidth. Such attacks are not merely hypothetical, but occur frequently in widely used P2P systems (e.g., [31, 34, 45, 39, 32, 5]). Fundamentally, what makes the problem difficult is that with today's network infrastructure, it is impossible for a third party to verify if a specific data transfer occurred between two colluding entities.

In this paper, we propose a *bandwidth puzzle* mechanism to make contribution-aware P2P content distribution systems robust to such collusion attacks. More specifically, in this mechanism, a *verifier* can confirm that claimed transfers of content actually occurred. For example, in content-streaming from a distinguished server through a tree of peers (e.g., [41, 15, 13]), or in P2P systems that incorporate a distinguished, trusted node for tracking content-transfer transactions (e.g., [44, 31, 40]), this distinguished node could additionally play the role of the verifier.

There are two key insights behind our design. First, to those peers (or "provers") claiming to have the file, the verifier presents puzzles for which the solution depends on the content of the file. More specifically, the solution is computationally simple for a prover who has the file, but more difficult for a prover who does not. In this respect, our puzzle design bears relationships to *proofs of data possession* (e.g., [6, 22]) and similar mechanisms; we detail the similarities and differences of our design in Section 2. Second, the verifier *simultaneously* presents these puzzles to all peers who currently claim to have the file, so as to make it difficult for a few peers who have the file to quickly solve both their own puzzles and puzzles for collaborators who do not. This simultaneity is a strategy borrowed from detectors for Sybil attacks [18, 29]; again, we detail the similarities and differences of our design in Section 2. The verifier checks the puzzle solutions and also notes the time taken by the provers to report the solutions. Any peer whose solution is incorrect or whose solution time is greater than a threshold θ is a suspect for engaging in fake transactions.

Our puzzle design is relatively simple and, due to its construction using only hash functions and pseudorandom functions, is efficient for both the verifier and for provers who do possess the files they claim. The security analysis of our design, however, is more subtle than its design might at first suggest. An analysis must account for any strategy by which adversaries might allocate portions of each puzzle's search space so as to optimally utilize the time θ that each has to invest

and, more importantly, the file bits that each possesses. We provide (in the random oracle model) a closed-form bound on the expected number of puzzles that a collection of adversaries can solve in θ time (using any such strategy), as a function of the number of hash computations that each adversary can perform in that time and the number of file bits each possesses. For example, this bound implies that for a file of size n, an instance of our puzzle construction ensures that all adversaries claiming to have a file must download an average of $\Omega(n/\log_2 n)$ file bits in order to solve their puzzles for this file in expectation. Moreover, this instance of our puzzle construction is very efficient: It enables the verifier to construct each puzzle in $\alpha \log_2 n$ pseudorandom function computations and two hash function computations, for some constant $\alpha > 1$, and to verify each puzzle in one comparison of hash function outputs. An honest prover must invest $O(n^\alpha/\log_2 n)$ time to solve this puzzle. We also provide a second, more refined bound on the expected number of puzzles that a collection of adversaries can solve in θ time (again in the random oracle model). While this bound is not a closed-form solution, it is efficiently computable and tighter, and thus applies to smaller file sizes.

We demonstrate the benefits of our bandwidth puzzles using simulations of the Maze P2P file-sharing system [44]. We choose Maze to guide our evaluation for two reasons. First, attacks of the type that we seek to defend against here have been documented in Maze [31], and so we can use these documented attacks to inform our evaluation. Second, Maze is a type of P2P system that can easily be adapted to use our bandwidth puzzles, owing to its structure utilizing a distinguished node to receive reports of which peers transferred files to which others and to reward peers accordingly. Our evaluations demonstrate that bandwidth puzzles prevent colluding attackers from benefiting in the Maze system by either reducing the number of attacker requests satisfied by up to 95% or by increasing the attackers' download time by up to 200%. Also, the puzzle scheme limits the impact of such attacks on legitimate client downloads, by providing honest peers with performance identical to the scenario when there are no attackers in the system.

To summarize, the contributions of this paper are: (i) The development and implementation of bandwidth puzzles (Section 4, Appendix B), a practical defense against a documented form of attack on P2P systems; (ii) analyses of our bandwidth puzzle construction (in the random oracle model) that bounds the success attainable by adversaries against it (Section 5, Appendix A); and (iii) an evaluation of our construction using simulations of the Maze P2P file-sharing system (Section 6).

2 Related Work

P2P and reputation systems: P2P protocols have been widely used for file distribution and multimedia streaming. While P2P systems offer tremendous scalability compared to single server systems, they suffer from the fundamental problem of managing incentives. Several works have demonstrated the limitations of P2P protocols in the presence of selfish or malicious users and free-riders [21, 39]. Measuring peer contributions have been suggested as mechanisms for mitigating these limitations (e.g., [41, 21]). Similarly, fair exchange and proof-of-service protocols (e.g. [30, 40]) ensure that a mutually distrusting sender and receiver can engage in a P2P transaction (i.e., guaranteeing that the sender receives credits if and only if the receiver receives the

file). However these existing mechanisms cannot prevent attackers from *freely* granting each other credits for fake transactions. Our mechanism limits fake transactions to ensure that attackers cannot achieve arbitrarily high rewards. Tit-for-tat contribution-awareness mechanisms such as those used in BitTorrent [2] are evidently not vulnerable to the kinds of collusion attacks we seek to mitigate in this paper. However, several studies (e.g, [21, 35, 28]) have pointed out the need to look beyond *pairwise* mechanisms such as tit-for-tat, and in particular make the case for designing more *global* contribution-aware mechanisms. Bandwidth puzzles serve as a basic building block for implementing robust P2P systems with global contribution-aware mechanisms.

Client puzzles: Client puzzles (e.g., [20, 25, 4, 19, 17, 43]) force clients to demonstrate a certain proof-of-work to a server before they can receive service. The goal of such puzzles is to throttle the number of requests clients can issue to the server to defend against spam and denial-of-service attacks. Our bandwidth puzzle scheme is an adaptation of this approach, in order to "throttle" the reward that a client can receive for claimed content transfers, by tying puzzle solving to content and issuing puzzle challenges simultaneously (see below).

Sybil attacks: Our adversary model involving colluding attackers claiming to have contributed more resources than they actually have is similar to the notion of a Sybil attack, which Douceur [18] suggests can be detected using simultaneous puzzle challenges. Levine et al. [29] provide a survey of known solutions to defend against Sybil attacks. These puzzles validate that each claimed Sybil "identity" possesses a certain minimum amount of computation resources. Bandwidth puzzles instead validate that collaborators expend a certain amount of communication resources, by leveraging the computational limits of each individual collaborator to force the collaborators to distribute puzzle solving and hence the file.

Proofs of data possession (PDP) and Proofs of retrievability (POR): Proofs of data possession (e.g., [6, 22, 7]) and proofs of retrievability (e.g., [26, 11, 38]) enable a client to verify that a remote store has not deleted or modified data the client stored there. There are several conceptual differences between the goals of a PDP/POR scheme and our puzzle scheme. First, PDP/POR schemes only focus on the interaction between a single prover and verifier, and do not deal with the case of colluding adversaries trying to claim credit for fake transactions. Second, PDP schemes minimize the communication interaction between the prover and the verifier, without specifically requiring that there be an asymmetry in the computation effort they expend. However, such an asymmetry (and the ability to tune that asymmetry) is crucial for our goals: the solving cost must be sufficiently high — even with the claimed content — to prevent one prover with the content from solving puzzles for many others, and at the same time puzzle generation and verification must be very efficient since the verifier must do these simultaneously for potentially many provers. One driving consideration in PDP/POR design is that the verifier no longer possesses the file about which it is querying. In contrast, we allow our verifier to possess the file, since the verifier can download it or might even be its origin (e.g., in a streaming scenario), and we leverage this to yield a design that is more efficient or flexible on certain axes than many PDP/POR designs. For example, most PDP schemes depend on cryptographic operations involving modular exponentiations (versus only hash functions and pseudorandom functions in our case), and existing POR constructions only allow a prespecified limited number of challenges. Also related is the work by Golle et al. [24] on communication enforcing signatures and storage enforcing commitment schemes. However,

their work provides a weaker guarantee than PDPs in that the prover expends some storage or communication cost proportional to the message size, but not necessarily the actual data.

3 System Model and Goals

We presume a system model consisting of a designated *verifier* and a collection of untrusted *peers*, also called *provers*. Any node can play the role of a verifier, provided that it can obtain the list of peers that purport to possess certain content at a point in time, and provided that it has access to that content (or to a peer who has the content and that it trusts). Peers transfer files between one another[1]; we are not concerned with the manner by which peers choose others for downloading. We require that peers are motivated to report to the verifier the files they claim to have downloaded from or uploaded to others, and consequently the files that each has. The goal of our mechanism is to enable the verifier to verify that the claimed bandwidth expenditures to transfer those files actually occurred.

The verifier does this by simultaneously presenting puzzles to the peers claiming to have a given file, and then recording the durations required by each prover to report its solution. We presume that the network latencies for transmitting puzzles and solutions between the verifier and the provers are sufficiently small that they do not contribute significantly to the puzzle solving time recorded by the verifier. On the basis of solution correctness and the puzzle-solving time that it records and compares to a threshold θ, the verifier generates a list of suspected colluders, i.e., peers suspected of not contributing the bandwidth claimed.

The puzzles presented by the verifier should have properties typical of puzzle schemes: (i) Provers should be unable to precompute puzzle solutions, or use previous puzzle solutions to generate new puzzle solutions. (ii) The verifier should incur low computational costs to generate puzzles and check puzzle solutions, and should incur low bandwidth costs to send the puzzles and receive the solutions. (iii) The verifier should be able to adjust the difficulty of the puzzle, as appropriate.

Unlike previous puzzle constructions, however, our bandwidth puzzles must additionally ensure that for colluding provers to all solve their puzzles within time θ, the file bits each receives in doing so, on average (and possibly before receipt of the puzzle itself), is roughly proportional to the size of the file. Were it not for the *simultaneity* in issuing puzzles, this would be impossible to achieve: each challenged prover could forward its puzzle to a designated solving prover who had the file, who could solve the puzzle and return it to the challenged prover. By (ii) above, both the puzzle and the solution would be small, implying that the bandwidth exchanged between the challenged prover and the solving prover would be small. Simultaneous puzzle challenges preclude such a strategy, since the solving prover is limited in the amount of work it can do (hence, the number of puzzles it can solve) in time θ.

The above goal comes with two caveats, however. First, without network monitoring support, it is not possible for the verifier to correctly ascertain which (if any) of multiple colluders actually

[1] In the interest of clarity, we will focus on a file-sharing model and present our discussion in terms of files. Note that this can be easily applied to media streaming deployments by considering a contiguous sequence of data chunks in the stream as a logical "file".

has the file, even if it detects one or more of them as colluders via our bandwidth puzzle scheme. For example, one prover with the file could invest its time in solving another prover's puzzle, even at the expense of solving its own. As such, the verifier detects provers who collude, but cannot ascertain who may or may not have the file. Second, to achieve the above goal, it is necessary that the file not be substantially compressible. If it were, then colluding provers could exchange the compressed version in lieu of the original file, and our goal could not be achieved. As such, in the remainder of this paper we treat the file as a random file, i.e., in which each bit is selected uniformly at random.

4 The Construction

We use " $\leftarrow$ " to denote assignment, and "$x \stackrel{R}{\leftarrow} X$" to denote the selection of an element from set X uniformly at random and its assignment to x. Concatenation is denoted by "||".

Security parameters: There are three security parameters that play a role in our construction. We use κ to denote the length of hash function outputs and keys to pseudorandom functions (see below). A reasonable value today might be $\kappa = 160$. The other two security parameters are denoted k and L, and together combine to dictate the difficulty of puzzle solving, and the costs that the verifier and prover incur in generating and solving puzzles, respectively.

Hash functions: Our construction employs two hash functions. $\mathsf{hash} : \{0,1\}^\kappa \times \{1,\ldots,L\} \times \{0,1\}^k \rightarrow \{0,1\}^\kappa$ is one such hash function. While hash functions typically take a single string as input, it is convenient to specify hash as taking three inputs (which can obviously be encoded in an unambiguous fashion as a string input to a one-input hash function such as SHA-1 [3]). In order to prove security of our construction in Section 5, we model hash as a random oracle [8]. The other hash function we use is $\mathsf{ans} : \{0,1\}^k \rightarrow \{0,1\}^\kappa$, though our proof does not require this to have the properties of a random oracle; e.g., collision-resistance (e.g., see [37]) suffices.[2]

Pseudorandom functions: A pseudorandom function family $\{f_K\}$ is a family of functions parameterized by a secret key $K \in \{0,1\}^\kappa$. Informally, the family has the property that it is infeasible to distinguish between an oracle for f_K where $K \stackrel{R}{\leftarrow} \{0,1\}^\kappa$, and an oracle for a perfectly random function with the same domain and range; see Goldreich et al. [23] for a formal definition. Our construction uses two pseudorandom function families, $\{f^1_K : \{1,\ldots,L\} \rightarrow \{0,1\}^\kappa\}$ and $\{f^2_K : \{1,\ldots,k\} \rightarrow \{1,\ldots,n\}\}$; we require that each f^2_K be injective, and thus that $k \leq n$. (Recall that n is the file size in bits.) A practical example of a pseudorandom function family is AES [1].

Construction: In our construction, a puzzle verifier generates puzzles with which to challenge a collection of provers simultaneously. Generally we will presume that the verifier generates one puzzle per prover, though there is no obstacle to sending multiple puzzles to each prover. Each puzzle consists of a hash value $\hat{h}$ output from hash and, intuitively, a collection of *index-sets* $I_1,\ldots,I_L$. Each index-set is a set of k random file indices, i.e., uniformly random samples from $\{1,\ldots,n\}$, without replacement. The verifier computes $\hat{h}$ as the hash of the file bits indexed by a randomly chosen index-set, appended together in an unambiguous order. Solving the puzzle means

[2] Minimizing reliance on random oracles is desirable, since they are not a standard cryptographic assumption [12].

finding which of the L index-sets has this property and, more specifically, the string that hashes to $\hat{h}$. Note that this requires at most L computations of hash for a prover who possesses the file, but could require substantially more for a prover who is missing some number of the file bits indexed by the index-sets in the puzzle.

This construction, as described, would be inefficient in a number of ways. First, for the verifier to transmit L index-sets of k indices each would require computation proportional to kL to generate the sets and then communication costs proportional to $kL \log_2 n$ to transmit them. To reduce these costs, the verifier generates index-sets pseudorandomly; see Figure 1. First, it randomly selects a key K_1 for the family f^1 and an index $\hat{\ell} \xleftarrow{R} \{1, \ldots, L\}$ to denote the index-set from which the challenge $\hat{h}$ will be generated. Second, it generates a key $\hat{K}_2 \leftarrow f^1_{K_1}(\hat{\ell})$ from which it generates index-set $I_{\hat{\ell}} = \{f^2_{\hat{K}_2}(1), \ldots, f^2_{\hat{K}_2}(k)\}$. Note that the verifier never needs to generate the other $L-1$ index-sets, reducing its computation to costs proportional to k alone. Simply sending K_1 and $\hat{h}$ suffices to enable the prover to search for $\hat{\ell}$, and incurs communication costs proportional only to κ. Because f^1 and f^2 are pseudorandom, the prover is unable to predict the index-sets better than random guessing prior to receiving K_1. Another way in which we reduce the communication costs in practice is to have the prover return $\mathsf{ans}(str)$ for the string str satisfying $\hat{h} = \mathsf{hash}(K_1, \hat{\ell}, str)$[3], rather than str itself. As we will see, it is generally necessary for k (and hence str) to grow as a function of n, whereas there is no such need for κ (the size of ans outputs).

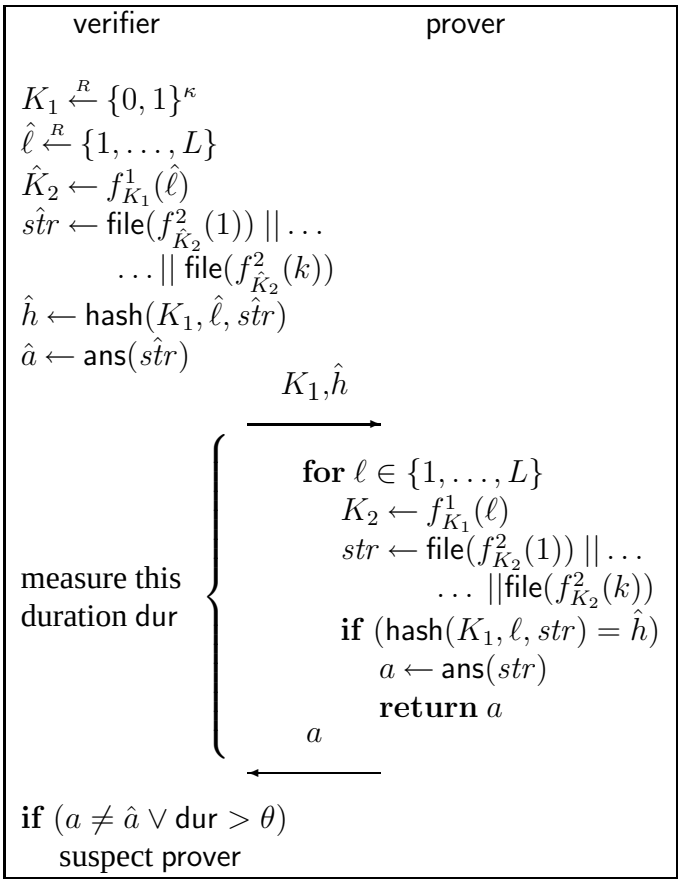


Figure 1: One bandwidth puzzle

5 Proof of Security

In order to prove the security of our construction, we first recap the properties we assume of the primitives we use. We assume that $\{f^1_K\}$ and $\{f^2_K\}$ are pseudorandom function families [23], and that ans is a collision-resistant hash function [37]. These primitives achieve their desired properties

[3]Including K_1 and $\hat{\ell}$ as inputs to hash ensures that the results of one puzzle-solving process cannot be used in the solving process of another puzzle, regardless of the file, k, and L.

— indistinguishability from a random function in the first case, and collision-resistance in the second — with all but negligible probability as a function of κ.[4] As such, in the remainder of this section, we simply assume that these properties hold, ignoring events that occur with probability negligible in κ.

Another primitive, namely hash, is modeled as a random oracle in our proof. Modeling hash in this way enables us to quantify the security of our scheme as a function of the number of hash computations. That is, we cap the number q_{hash} of hash queries that any prover can complete in θ time, and then quantify the probability with which the prover returns $\hat{a}$ as a function of q_{hash}. Moreover, modeling hash as a random oracle enables us to exploit the property in our proof that one such computation provides no information about the computation of hash on any other value.

Of course, the probability that an adversarial prover succeeds in returning $\hat{a}$ within θ time (i.e., after making at most q_{hash} queries to hash) also depends on the number of file bits it receives before and during the puzzle-solving process. To model the receipt of file bits in our proof, we also find it convenient to model a prover's retrieval of file bits as calls to a random oracle file : $\{1, \ldots, n\} \rightarrow \{0, 1\}$. As discussed in Section 3, our construction requires that the file being exchanged have sufficient empirical entropy to be incompressible, as otherwise adversaries could "defeat" our verification by exchanging (in full) the compressed file. Consequently, we model the file as a random string of length n, and track the number of file bits that an adversary retrieves prior to returning a puzzle solution by the number of queries it makes to its file oracle.

5.1 A closed-form bound

In this section we present a closed-form bound for adversaries' ability to solve puzzles:

Theorem 5.1. *Let* hash *and* file *be random oracles. Consider A adversaries working in collaboration, each permitted q_{hash} queries to* hash, *collectively permitted Aq_{file} queries to* file, *and collectively challenged to solve a set $\mathcal{P}$ of P puzzles. For any $k \geq \log_2(q_{\mathsf{hash}}/P + L) + 2$ and $\delta > 0$, the expected number of puzzles that these adversaries can solve collectively is at most*

$$\frac{AP^2(1+\delta)kq_{\mathsf{file}}}{n(k - \log_2(q_{\mathsf{hash}}/P + L) - 1)} + \frac{AP}{L} + Pn\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{PkL/n} \tag{1}$$

To see an example of Theorem 5.1 when, say, sending one puzzle to each prover and $q_{\mathsf{hash}} = L$, consider setting $L = n^{\alpha}/k$ for some $\alpha > 1$ and $k = \alpha \log_2 n$, which is at least $\log_2(q_{\mathsf{hash}}/P + L) + 2$ (a requirement of Theorem 5.1) for $n \geq 256$. Then, by instantiating (1) with these values and simplifying, the number of puzzles that A adversaries can solve in expectation, out of the $P = A$ that they receive, is at most

$$\alpha(1+\delta)\left(\frac{A^3}{\log_2 A}\right)\left(\frac{\log_2 n}{n}\right)q_{\mathsf{file}} + A^2\left(\frac{\alpha \log_2 n}{n^{\alpha}}\right) + An\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{An^{\alpha-1}}$$

[4] A function $g(\cdot)$ is *negligible* if for any positive polynomial $p(\cdot)$, there is a κ_0 such that $g(\kappa) \leq 1/p(\kappa)$ for all $\kappa \geq \kappa_0$.

for $n \geq 256$. Treating A and q_{file} as constants, note that each of the three terms in this sum goes to zero as $n \to \infty$. Moreover, in order for the A adversaries to solve their A puzzles in expectation, we can solve for q_{file}:

$$q_{\mathsf{file}} \geq \frac{1}{\alpha(1+\delta)} \left(\frac{\log_2 A}{A^2}\right) \left(\frac{n}{\log_2 n}\right) \left(1 - A\left(\frac{\alpha \log_2 n}{n^\alpha}\right) - n\left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{An^{\alpha-1}}\right) = \Omega\left(\frac{n}{\log_2 n}\right)$$

Though we defer the full proof of Theorem 5.1 to Appendix A, we next present one part of that proof, as our tighter bound in Section 5.2 builds from it. This result is stated in terms of the experiment in Figure 2 for a single adversary $\mathcal{A}$. Aside from the random oracles file and hash, this experiment exactly tracks the verifier's actions in the protocol in Figure 1. In the experiment, let $\mathbf{Func}(\mathsf{Dom} \to \mathsf{Rng})$ denote the set of all functions with domain Dom and range Rng.

$\mathbf{Expt}(\mathcal{A})$:
- $\mathsf{file} \xleftarrow{R} \mathbf{Func}(\{1,\ldots,n\} \to \{0,1\})$
- $\mathsf{hash} \xleftarrow{R} \mathbf{Func}(\{0,1\}^\kappa \times \{1,\ldots,L\} \times \{0,1\}^k \to \{0,1\}^\kappa)$
- $K_1 \xleftarrow{R} \{0,1\}^\kappa$
- $\hat{\ell} \xleftarrow{R} \{1,\ldots,L\}$
- $\hat{K}_2 \leftarrow f^1_{K_1}(\hat{\ell})$
- $\hat{str} \leftarrow \mathsf{file}(f^2_{\hat{K}_2}(1)) || \ldots || \mathsf{file}(f^2_{\hat{K}_2}(k))$
- $\hat{h} \leftarrow \mathsf{hash}(K_1, \hat{\ell}, \hat{str})$
- $\hat{a} \leftarrow \mathsf{ans}(\hat{str})$
- $a \leftarrow \mathcal{A}^{\mathsf{file},\mathsf{hash}}(K_1, \hat{h})$
- **if** $(a = \hat{a})$
  - **return** 1
- **return** 0

Figure 2: Experiment for Theorem 5.2

In the proofs of Theorems 5.1 and 5.2, a property of a puzzle that influences how easy it is for adversaries to solve is how "spread out" the indices are that comprise its index-sets. Thus, we define the event $\mathsf{Spread}(I, s)$, where I is a multiset (a set allowing repetition of elements) of indices from $\{1,\ldots,n\}$ and s is an integer, to denote that no $i \in \{1,\ldots,n\}$ appears s or more times in I.

Theorem 5.2. *Let* file *and* hash *be random oracles, and let $\mathcal{A}$ be an adversary making q_{file} queries to* file *and q_{hash} queries to* hash. *For any $s \geq 1$ and any $k \geq \log_2(q_{\mathsf{hash}} + L) + 2$,*

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{Spread}(I, s)\right] \leq \frac{1}{L}\left(\frac{sq_{\mathsf{file}}}{k - \log_2(q_{\mathsf{hash}} + L) - 1} + 1\right)$$

where I is the multiset $I = \bigcup_{\ell=1}^{L} I_\ell$ and I_ℓ is the set $I_\ell = \{f^2_{K_2}(1), \ldots, f^2_{K_2}(k)\}$ for $K_2 = f^1_{K_1}(\ell)$.

Proof.

$$\begin{aligned}\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{Spread}\,(I, s)\right] &= \sum_{\ell=1}^{L} \mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \mathsf{P}\left[\hat{\ell} = \ell \mid \mathsf{Spread}\,(I, s)\right] \\ &= \frac{1}{L} \sum_{\ell=1}^{L} \mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \qquad (2)\end{aligned}$$

We now focus on bounding $\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{Spread}(I, s)\right]$ from above. Let $\mathsf{confirm}$ be the event that the $\mathcal{A}$ performs a query to hash that returns the challenge value $\hat{h}$, within the q_{hash} oracle queries available to it. Then,

$$\begin{aligned}
&\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \\
&= \mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{confirm} \wedge \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \mathsf{P}\left[\mathsf{confirm} \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \; + \\
&\quad \mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \neg\mathsf{confirm} \wedge \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \mathsf{P}\left[\neg\mathsf{confirm} \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \qquad (3)
\end{aligned}$$

Let y_ℓ be a random variable denoting the number of queries of the form $\mathsf{hash}(K_1, \ell, *)$ that $\mathcal{A}$ makes in an execution. Let $w_{\ell i}$ be a binary random variable such that $w_{\ell i} = 1$ if $i \in I_\ell$ and $\mathcal{A}$ queries $\mathsf{file}(i)$, and $w_{\ell i} = 0$ otherwise. Let $w_\ell = \sum_{i=1}^{n} w_{\ell i}$. We now take

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{confirm} \wedge \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \;\leq\; 1 \qquad (4)$$

$$\mathsf{P}\left[\mathsf{confirm} \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \;\leq\; \frac{y_\ell}{2^{k-w_\ell}} \qquad (5)$$

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \neg\mathsf{confirm} \wedge \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \;\leq\; \frac{1}{2^{k-w_\ell} - y_\ell} \qquad (6)$$

$$\mathsf{P}\left[\neg\mathsf{confirm} \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \;\leq\; \frac{2^{k-w_\ell} - y_\ell}{2^{k-w_\ell}} \qquad (7)$$

(5) and (7) follow from $\mathcal{A}$ querying $\mathsf{hash}(K_1, \ell, str)$ for only y_ℓ values str of the 2^{k-w_ℓ} such possible values for the $k - w_\ell$ bits it did not retrieve from file. In the event $\neg\mathsf{confirm}$, the probability that $\mathcal{A}$ produces $\hat{a}$ is simply that with which it guesses correctly from the remaining $2^{k-w_\ell} - y_\ell$ values and submits this str to ans, leading to (6). Plugging these into (3), we get

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{Spread}\,(I, s)\right] \;\leq\; \min\left\{\frac{y_\ell + 1}{2^{k-w_\ell}}, 1\right\}$$

and then plugging this into (2), we get

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{Spread}\,(I, s)\right] \;\leq\; \frac{1}{L}\sum_{\ell=1}^{L} \min\left\{\frac{y_\ell + 1}{2^{k-w_\ell}}, 1\right\} \;=\; \frac{1}{L2^k}\sum_{\ell=1}^{L} \min\left\{(y_\ell + 1)2^{w_\ell}, 2^k\right\} \qquad (8)$$

Consequently, we now focus on bounding

$$\sum_{\ell=1}^{L} \min\left\{y'_\ell 2^{w_\ell}, 2^k\right\} \qquad (9)$$

from above where $y'_\ell = y_\ell + 1$, subject to the constraints

$$\sum_{\ell=1}^{L} y'_\ell \leq q_{\mathsf{hash}} + L \qquad (10) \qquad\qquad \sum_{\ell=1}^{L} w_\ell \leq s q_{\mathsf{file}} \qquad (11)$$

where (10) follows from $\sum_{\ell=1}^{L} y_\ell \leq q_{\mathsf{hash}}$. To do so, we first note that for any fixed $w_1, \ldots, w_L$, a choice of $y'_1, \ldots, y'_L$ that maximizes (9) is one that maximizes y'_ℓ for the largest values w_ℓ. That is, if we order $w_1, \ldots, w_L$ in nonincreasing order, then setting $y'_\ell = 2^{k-w_\ell}$ for $\ell = 1, 2, \ldots, m$ where

$$\sum_{\ell=1}^{m} 2^{k-w_\ell} \leq q_{\mathsf{hash}} + L < \sum_{\ell=1}^{m+1} 2^{k-w_\ell} \tag{12}$$

and setting $y'_{m+1} = q_{\mathsf{hash}} + L - \sum_{\ell=1}^{m} 2^{k-w_\ell}$ maximizes (9), and the maximum value for (9) is then

$$\sum_{\ell=1}^{L} \min\left\{y'_\ell 2^{w_\ell}, 2^k\right\} < (m+1)2^k \tag{13}$$

Consequently, to obtain bounds on the maximum value of (9) for a given q_{file} and q_{hash}, it suffices to find $w_1, \ldots, w_L$ so as to maximize m subject to (11) and (12). For any fixed m, $\sum_{\ell=1}^{m} 2^{-w_\ell}$ is minimized by setting $w_\ell = \lceil sq_{\mathsf{file}}/m \rceil$ for $1 \leq \ell \leq (sq_{\mathsf{file}} \bmod m)$ and $w_\ell = \lfloor sq_{\mathsf{file}}/m \rfloor$ for $(sq_{\mathsf{file}} \bmod m) + 1 \leq \ell \leq m$. As such, the maximum value of m is

$$\arg\min_{m>0} \begin{cases} q_{\mathsf{hash}} + L - m2^{k-(sq_{\mathsf{file}}/m)} & \text{if } m \mid sq_{\mathsf{file}} \\ q_{\mathsf{hash}} + L - (2m - (sq_{\mathsf{file}} \bmod m))2^{k-\lceil sq_{\mathsf{file}}/m \rceil} & \text{otherwise} \end{cases}$$

If the maximum such m divides sq_{file}, then $m2^{k-(sq_{\mathsf{file}}/m)} \leq q_{\mathsf{hash}} + L$ implies $m \leq sq_{\mathsf{file}}/(k - \log_2(q_{\mathsf{hash}} + L))$, and otherwise $m \leq sq_{\mathsf{file}}/(k - \log_2(q_{\mathsf{hash}} + L) - 1)$. Combining this with (13), we get that

$$\sum_{\ell=1}^{L} \min\left\{y'_\ell 2^{w_\ell}, 2^k\right\} < (m+1)2^k \leq \left(\frac{sq_{\mathsf{file}}}{k - \log_2(q_{\mathsf{hash}} + L) - 1} + 1\right) 2^k$$

and so the result follows by combining this with (8). □

5.2 A tighter, computable bound

We now provide an efficiently computable (albeit non-closed-form) bound that can be used in place of that in Theorem 5.1. We first revisit the proof of Theorem 5.2 to get a bound for $\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1\right]$. First, note that (4)–(8) do not depend on $\mathsf{Spread}\,(I, s)$, and thus we can restate (8) as:

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1\right] \;\leq\; \frac{1}{L2^k} \sum_{\ell=1}^{L} \min\left\{(y_\ell + 1)2^{w_\ell}, 2^k\right\} \tag{14}$$

Now, define a binary variable d_i that takes the value 1 if $\mathcal{A}$ makes a query to the file for index i and 0 otherwise. Also, let o_i denote the number of times index i appears in I. As in the proof of Theorem 5.2, we focus on bounding (9) from above, subject to the constraints

$$\sum_{\ell=1}^{L} y'_\ell \leq q_{\mathsf{hash}} + L \tag{15}$$

$$\sum_{\ell=1}^{L} w_\ell = \sum_{i=1}^{n} o_i d_i \tag{16}$$

The only difference between the analysis here and the analysis of Theorem 5.2 is between (11) and (16). Specifically, we count $\sum_{\ell=1}^{L} w_\ell$ exactly in terms of o_i and d_i instead of using the (loose) upper bound sq_{file}. The rest of the analysis follows identically except that we replace the term sq_{file} with $\sum_{i=1}^{n} o_i d_i$. With this, we now have

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1\right] \leq \frac{1}{L}\left(\frac{\sum_i^n o_i d_i}{k - \log_2(q_{\mathsf{hash}} + L) - 1} + 1\right) \tag{17}$$

We can now extend the result in (17) to the scenario of Theorem 5.1, where A adversaries are working in collaboration, each permitted q_{hash} queries to hash, collectively permitted Aq_{file} queries to file, and collectively challenged to solve a set $\mathcal{P}$ of P puzzles. As in the proof of Theorem 5.1 (see Appendix A), index the adversaries by $1, \ldots, A$ and the puzzles by $1, \ldots, P$. Let S^{ap} be a binary random variable such that $S^{ap} = 1$ if adversary a produces the solution for puzzle p, and $S^{ap} = 0$ otherwise. Letting $S = \sum_{a=1}^{A}\sum_{p=1}^{P} S^{ap}$, we want to bound $\mathsf{E}\left[S\right]$ from above.

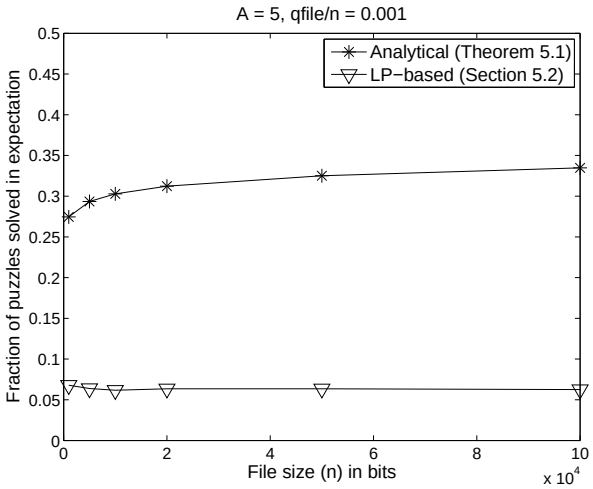


Figure 3: Example for bounds of Section 5

Now, define a binary variable d_{ai} that takes the value 1 if adversary a makes a query to the file for index i and 0 otherwise. Also, let o_{pi} denote the number of times index i appears in puzzle p. Let q^{ap}_{hash} denote the number of queries of the form $\mathsf{hash}(K_1^p, *, *)$ made by adversary a, and let q^{a}_{file} denote the number of queries by adversary a to file. Under the constraints

$$q^a_{\mathsf{file}} = \sum_{i=1}^{n} d_{ai} \text{ for } 1 \leq a \leq A \tag{18}$$

$$\sum_{a=1}^{A} q^a_{\mathsf{file}} \leq Aq_{\mathsf{file}} \tag{19}$$

$$\sum_{p=1}^{P} q^{ap}_{\mathsf{hash}} \leq q_{\mathsf{hash}} \text{ for } 1 \leq a \leq A \tag{20}$$

we now seek to bound

$$\sum_{a=1}^{A}\sum_{p=1}^{P} \mathsf{E}\left[S^{ap}\right] = \sum_{a=1}^{A}\sum_{p=1}^{P} \frac{1}{L}\left(\frac{\sum_i^n o_{pi} d_{ai}}{k - \log_2(q^{ap}_{\mathsf{hash}} + L) - 1} + 1\right) \leq \sum_{a=1}^{A}\sum_{p=1}^{P} \frac{1}{L}\left(\frac{\sum_i^n o_{pi} d_{ai}}{k - \log_2(q_{\mathsf{hash}} + L) - 1} + 1\right) \tag{21}$$

The second equality just applies (17) to each puzzle-adversary combination, and the inequality follows since we are only decreasing the denominator by using q_{hash} instead of q^{ap}_{hash}.

Now, consider the optimization problem defined by the objective function (21) subject to the constraints (18)–(20). For any fixed $\mathcal{P}$, this optimization problem is an integer linear program with binary variables d_{ai}. While it is hard to solve such large integer linear programs, relaxing the integer constraints on the variables (and bounding them to be fractional variables in the range $[0, 1]$) can only increase the optimum value and renders the problem efficiently solvable. Thus, by solving this linear program we get an upper bound on the expected number of puzzles solved.

An example of the benefit of this approach to bounding the expected number of puzzles that A adversaries can together solve, versus the closed-form bound offered in Theorem 5.1, is shown in Figure 3. This figure plots the bound of Theorem 5.1 and the bound computed via the method detailed in this section, for parameter settings $\alpha = 1.5$, $k = \alpha \log_2 n$, $L = n^\alpha / \log_2 n$, $q_{\mathsf{file}}/n = 0.001$, $q_{\mathsf{hash}} = L$, and $A = P = 5$. As we can see, the bound in this section achieves roughly a factor of 4–5 improvement over the bound of Theorem 5.1 for these choices of parameters.

6 Benefits in a P2P File-Sharing System

We evaluate the practical system impact of the bandwidth puzzle scheme in the context of the Maze [44, 31] P2P file sharing system. Our choice of Maze was motivated by two factors. First, the Maze system uses a centralized authority for auditing peer contributions. This is a natural choice to serve as the verifier in charge of issuing and verifying puzzles. Second, the Maze system incentivizes peers to upload content based on a "points" system that we describe briefly below. Such incentive mechanisms are important for the viability of P2P systems to encourage uploaders. However, a subsequent measurement study demonstrated a wide range of collusion attacks [31]. Our goal is to minimize the impact of collusion in such systems using bandwidth puzzles.

Maze system model: The Maze system has a centralized server to which every peer authenticates and actively reports the set of files it possesses. In addition to providing query and search functions, the server maintains a "points system". Peers consume points while downloading files and earn points while uploading files. Uploads add more points than downloads consume: each peer earns 1.5 points for every 1MB of content uploaded, but consumes less than 1 point per MB of download. (The actual number of points consumed is a function of file size, but in our evaluation we use a simplified model assuming a fixed consumption rate of 1 point per MB). New users are bootstrapped with a prespecified number of points allowing some initial "free" downloads before they can contribute. The system incentivizes peers to upload content by prioritizing download requests based on the number of points the requesting peer currently has. (Specifically, requests are queued with priority $rqsttime - 3 \log \rho$, where ρ is the current number of points the requester has.) In addition, the system rate limits the upload rate to peers with less than 512 points.

Adding bandwidth puzzles to Maze: In the traditional Maze system, upon receiving a report of a completed file exchange, the server subtracts points from the downloader and credits points to the uploader. With a system augmented with bandwidth puzzles, handling transactions is slightly different. The server debits points from the downloader's account as before. However, it does not immediately credit the uploader for the transaction. Instead, it records a *pending transaction* identified by the 4-tuple: $\langle uploader, downloader, filename, credits \rangle$.

The server sends puzzle challenges, in the role of the verifier. We refer to the time between puzzle challenges as a *puzzle epoch*, and assume that the duration of a puzzle epoch is significantly larger than the per-puzzle timeout θ. At the start of each puzzle epoch, for each file for which an exchange was reported during the previous puzzle epoch, the server retrieves the set of all peers that currently claim to have the file and generates puzzles for these clients. One subtle issue here is in setting the puzzle timeout for each client: some clients might receive multiple puzzles during a single puzzle epoch since they might claim to possess more than one file. Thus, the

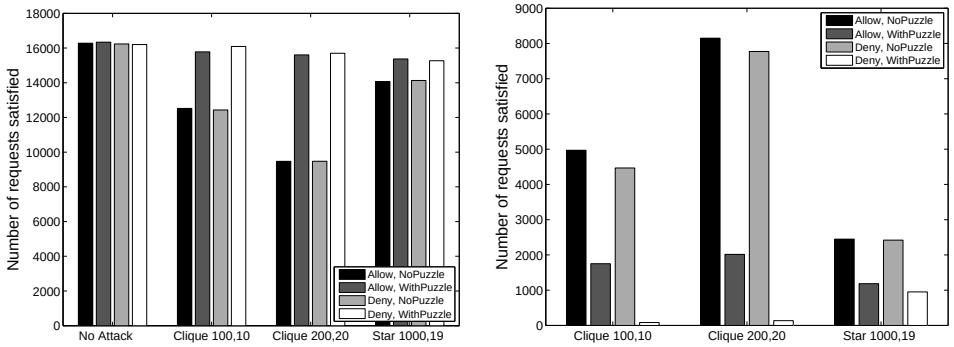
timeout for a puzzle should depend on the total number of puzzles sent to that client in this epoch. Specifically, for client x, the p-th puzzle is issued with threshold $\theta_p = (p-1) \times \theta$, where θ is the per-puzzle duration threshold. (We assume that legitimate clients prioritize their received puzzles based on the timeout indicated in the puzzle descriptor.) On receiving a response for a puzzle sent to x for file $filename$, the server verifies if the answer is correct and if the response came within the puzzle-specific timeout. If so, the server *validates* all pending transactions of the form $\langle uploader = *, x, filename, credits = * \rangle$, thereby crediting the uploaders in these transactions with the corresponding credits.

Attack Model: We specify attacks by a *collusion graph*. A collusion graph is a directed graph, where each vertex is a malicious peer (either an actual or Sybil node). An edge $x \rightarrow y$ represents an *fake uploader* relationship wherein peer x reports "fake" transactions to the server on behalf of peer y. In other words, x requests the server to credit y for uploading a file, even though y does not actually spend any bandwidth for the transfer. In our model, each such x will periodically report fake transactions to the server in addition to its actual (legitimate) transactions.

The notion of a collusion graph is general enough to capture different collusion patterns. For example, in the Maze measurement study [31], the authors find that most collusion patterns comprise two or three mutually colluding nodes. This is represented as a directed *clique*. Other forms of observed collusion patterns include *star* topologies (similar to a Sybil attack). In a *star* topology, the only role of the Sybil identities is to grant points to their master node and they do not generate any real file requests. In our evaluation, we focus on two types of graphs: *cliques* and *stars*.

Simulation Framework: We implemented an event-driven simulator (three thousand lines of C++ code) to model file exchanges, transaction reports, and puzzle challenges. We make some simplifying assumptions to make our simulation framework scalable and tractable. In particular, we do not model network congestion effects, and instead assume that the only bandwidth bottleneck is the upstream bandwidth bottleneck of the peers [9]. Also, we assume that all files are split into 1MB chunks, and that all file exchanges and transaction reports happen at the granularity (or an integral multiple) of this chunk size. To simplify the request queue dynamics, each peer queues file transfer requests based on the downloader's points (using the same prioritization as in Maze) and serves one at a time, without preemption.

Our simulation framework differs from the actual Maze system in a few ways. First, we assume that peers request files only via the server whereas in the Maze system, peers can also "browse" and request their "friends"' files. This assumption does not impact our results; it merely represents an alternative file request pattern. Second, the Maze system assumes a single uploader for each file exchange. In contrast, we assume that files are split into chunks (similar to BitTorrent) and each peer can request different chunks of the same file from multiple available uploaders. Again, this has no impact on the strength of the collusion attacks, since we assume that the attackers' fake transactions deviate from this assumption (fake transactions credit a single colluding partner with the entire upload). Third, Maze serves requests from persistent "free-riders" (clients with scores lower than a prespecified threshold) with lower upstream bandwidth compared to other requests. We do not model this, to avoid undesirable effects arising from the fact that each peer uploads to one peer at a time. Instead, we evaluate two configurations: one in which peers allow free-riders (but give their requests very low priority), and another in which peers deny service to free-riders.



(a) Legitimate clients (b) Attackers

Figure 4: Number of requests satisfied. Each bar represents one of four Maze configurations. We can either "Allow" or "Deny" free-riders and can choose to implement/not implement bandwidth puzzles. Each cluster represents a specific attack configuration.

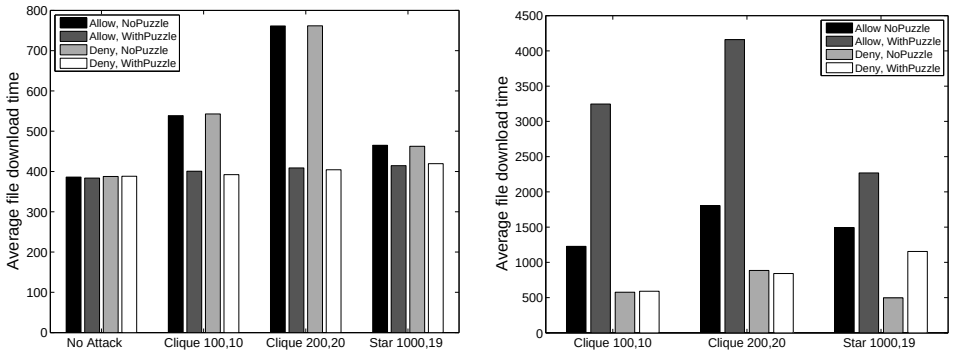
Each of our simulations runs for 10^5 units of simulation time where each unit of simulation time corresponds to 100ms of real time. There are 100 files shared with file request popularity following a Zipf distribution. The simulation consists of 1000 legitimate clients. Each legitimate client chooses an arrival time uniformly at random in $[0, 10^5]$ and has an average lifetime of $20,000$ units. Each client is bootstrapped on arrival with an initial set of files. Attackers arrive at time $20,000$ and persist until the end of the simulation. For each result, we repeat the simulation over 5 random initial seeds and present the averages across the multiple runs.

To model attackers' responses to puzzle challenges, we assume that a puzzle sent to a peer who does not have the file (or a fake peer) is solved with a fixed probability 0.1. We specify attacks by a graph, e.g., Clique(100,10) denotes that there are 100 attackers organized in colluding cliques each of size 10. Similarly, Star(1000,19) implies that there are 1000 attackers in total, organized in 50 star graphs each with 19 leaf nodes representing the fake (Sybil) identities. In our experiments, we evaluate three attack scenarios: Clique(100,10), Clique(200,20), and Star(1000,19). We focus on collusions of moderate size since the Maze measurement study [31] found that most collusion patterns involved a small number of attackers.[5]

Results: We focus on two metrics in our simulations: number of requests satisfied and the average request completion time. In each case, we measure the metric for both legitimate clients and for attackers. Attackers impact the performance of legitimate clients in two ways. First, each attacker request served decreases the total bandwidth available to legitimate client requests. Second, attackers can get better, faster service by boosting their points via fake transactions. This means that requests from legitimate clients may end up with lower priority. The goal of the bandwidth puzzle scheme in the context of Maze is to ensure that attackers do not degrade the performance of legitimate clients and attackers do not receive undue advantage from fake transactions.

Figure 4(a) shows the number of legitimate clients' requests satisfied. Bandwidth puzzles boost the performance by 11-70%. The benefit is slightly better when free-rider requests are not serviced. Also, across Clique(100,10) and Clique(200,20), as the number of attackers increases,

[5]Additionally, in some content-distribution systems (e.g., tree-based streaming), there are bounds on the number of peers that any peer can interact with, thus inherently limiting the size of collusion attacks that can be launched.



(a) Legitimate clients (b) Attackers

Figure 5: Mean file download time

the benefit provided by the puzzle mechanism increases two-fold. In Figure 4(b) we see that bandwidth puzzles decrease the total number of attackers' requests satisfied by 50-75% in the case when free-riders are allowed, and by 60-95% when free-rider requests are denied.[6]

Figures 5(a) and 5(b) show the average completion time for legitimate and attacker requests. Bandwidth puzzles improve the mean download time of legitimate client requests by 12-50%. Figure 5(b) shows that in the allow free-rider configuration, the mean download time for attacker requests increases by 60-200%. Surprisingly, for some deny free-rider configurations, the average attacker completion time is lower when bandwidth puzzles are used. This anomaly can be explained by referring back to Figure 4(b) and the Maze system design. Recall that in Maze, each peer is given some initial credits that it can use to download files. In the case of the deny free-rider configuration (with bandwidth puzzles), the only attacker requests satisfied correspond to these initial *free* downloads. Since attackers have credits initially, these requests see smaller queueing delays.

For reference, we also show the results when there are no attackers in Figures 4(a) and 5(a). When bandwidth puzzles are used, the number of legitimate client requests satisfied and the mean download time with and without the attack are almost identical. This shows that attackers have little or no impact on the performance of legitimate clients in a system with bandwidth puzzles.

7 Discussion

Choosing θ: In a real-world deployment, provers may have heterogeneous computational capabilities. Thus, θ should be set such that the slowest machine can solve bandwidth puzzle within θ time. However, choosing too large a θ (or alternatively a determined adversary running a large server farm for solving puzzles) opens the possibility of allowing one adversary on a fast machine to solve multiple puzzles. This is not a specific limitation of our bandwidth puzzle scheme; this is a broader limitation of client puzzle schemes. Puzzle schemes are most effective when attackers have difficulty in gaining access to vast computing resources and when the disparity between

[6]It may appear anomalous that Star(1000,19) has fewer attacker requests satisfied even though the total number of attackers is greater. However, recall that the Star(1000,19) attack has only 50 active nodes generating file requests. The fake identities are passive peers and do not generate any requests.

verifiers is not too large. Memory-bound puzzles [4, 19, 17] can help mitigate moderate levels of heterogeneity. We plan to explore if our bandwidth puzzle scheme can be extended to possess such properties.

Ease of deployment: Our experience with extending a Maze-like system to incorporate bandwidth puzzles (Section 6) required only a few hundred lines of additional code to the server and client implementations. This suggests that the overhead to modify systems that have a distinguished trusted node and have access to the content (e.g., [41, 15, 13, 44, 31, 40]) is minimal. For scenarios in which it is not feasible to have a central authority, we plan to explore more distributed architectures (e.g., [35, 42]) in future work.

8 Conclusions

Peer-to-peer systems have long been plagued by the problem of malicious or selfish adversaries exploiting weaknesses in the underlying incentive and reputation mechanisms. In particular, it has been observed that a group of colluding adversaries can implement a "ballot stuffing" attack (i.e., by reporting having received service from one another without spending any actual resources) to enhance their reputations to get preferential service.

Our work provides a simple, yet powerful primitive to thwart such collusion attacks in P2P systems. It is based on the key insight of simultaneously challenging the adversaries with *bandwidth puzzles* to demonstrate that the purported data transfers actually took place. We provide a security analysis of our scheme in the random oracle model (both a closed-form and a tighter, non-closed-form bound). We also demonstrate the practical impact of our scheme in the context of a P2P file-sharing system via simulation experiments. Our experiments demonstrate that the bandwidth puzzles prevent colluding attackers from gaining undue advantage via ballot stuffing attacks and from impacting the performance of honest peers.

References

[1] Advanced encryption standard. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[2] Bittorrent. `http://www.bittorrent.com`.

[3] Secure hash standard. `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.

[4] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. *ACM Transactions on Internet Technology*, 5:299–327, 2005.

[5] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5, 2000.

[6] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proc. of ACM CCS*, 2007.

[7] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and Efficient Provable Data Possession. IACR eArchive 2008/114 at http://eprint.iacr.org/2008/114.pdf, 2008.

[8] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, Nov. 1993.

[9] A. Bharambe, C. Herley, and V. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *Proceedings of IEEE INFOCOM*, 2006.

[10] R. Bhattacharjee and A. Goel. Avoiding Ballot Stuffing in eBay-like Reputation Systems. In *Proc. of ACM SIGCOMM P2P-ECON*, 2005.

[11] K. Bowers, A. Juels, and A. Oprea. Proofs of Retrievability: Theory and Implementation. IACR eArchive 2008/175 at http://eprint.iacr.org/2008/175.pdf, 2008.

[12] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proceedings of the 30th ACM Symposium on Theory of Computing*, pages 209–218, May 1998.

[13] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proc. of ACM SOSP*, 2003.

[14] K. Cho, K. Fukuda, and H. Esaki. The impact and implications of the growth in residential user-to-user traffic. In *Proc. ACM SIGCOMM*, 2006.

[15] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proc. of ACM SIGMETRICS*, 2000.

[16] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the ACM Conference on Electronic Commerce*, 2000.

[17] S. Doshi, F. Monrose, and A. Rubin. Efficient memory bound puzzles using pattern databases. In *Proceedings of the International Conference on Applied Cryptography and Network Security*, 2006.

[18] J. Douceur. The Sybil attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Mar. 2002.

[19] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Proceedings of CRYPTO 2003*, Aug. 2003.

[20] C. Dwork and M. Naor. Pricing via processing, or, combatting junk mail. In *Advances in Cryptology – CRYPTO '92 (Lecture Notes in Computer Science 740)*, pages 139–147, 1993.

[21] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *Proc. of ACM E-Commerce Conference*, 2004.

[22] D. L. G. Filho and P. S. L. M. Barreto. Demonstrating data possession and uncheatable data transfer. IACR eArchive 2006/150 at `http://eprint.iacr.org/2006/150.pdf`, 2006.

[23] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1984.

[24] P. Golle, S. Jarecki, and I. Mironov. Cryptographic primitives enforcing communication and storage complexity. In *Proc. of Financial Cryptography*, 2002.

[25] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of the 6th ISOC Network and Distributed System Security Symposium*, Feb. 1999.

[26] A. Juels and B. S. K. Jr. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, Oct. 2007.

[27] K. Kong and D. Ghosal. Mitigating server-side congestion in the Internet through pseudoserving. *IEEE Transactions on Networking*, 7(4):530–545, Aug. 1999.

[28] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In *Proc. of Workshop on Economics of Peer-to-Peer Systems*, 2004.

[29] B. N. Levine, C. Shields, and N. B. Margolin. A survey of solutions to the sybil attack. Technical Report 2006-052, University of Massachusetts Amherst, Oct. 2006.

[30] J. Li and X. Kang. Proof of service in a hybrid P2P environment. In *Proc. of ISPA Workshops*, 2005.

[31] Q. Lian, Z. Zhang, M. Yang, B. Y. Zhao, Y. Dai, and X. Li. An empirical study of collusion behavior in the Maze P2P file-sharing system. In *Proceedings of the 2007 International Conference on Distributed Computting Systems*, June 2007.

[32] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting BitTorrent for fun (but not profit). In *Proceedings of the Fifth International Workshop on Peer-to-Peer Systems*, Feb. 2006.

[33] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[34] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2007.

[35] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One hop reputations for peer to peer file sharing workloads. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, Apr. 2008.

[36] D. Purandare and R. Guha. BEAM: An Efficient Framework for Media Streaming. In *Proc. of IEEE LCN*, 2006.

[37] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision-resistance. In *Fast Software Encryption, 11th International Workshop, FSE 2004 (Lecture Notes in Computer Science 3017)*, pages 371–388, 2004.

[38] H. Shacham and B. Waters. Compact Proofs of Retrievability. IACR eArchive 2008/073 at http://eprint.iacr.org/2008/073.pdf, 2008.

[39] M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free-riding in BitTorrent networks with the large view exploit. In *Proc. of IPTPS*, 2007.

[40] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki. Dandelion: Cooperative Content Distribution with Robust Incentives. In *Proc. of USENIX ATC*, 2007.

[41] Y. Sung, M. Bishop, and S. Rao. Enabling Contribution Awareness in an Overlay Broadcasting System. In *Proc. ACM SIGCOMM*, 2006.

[42] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: a secure economic framework for P2P resource sharing. In *P2P Econ*, 2003.

[43] X. Wang and M. K. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.

[44] M. Yang, H. Chen, B. Y. Zhao, Y. Dai, and Z. Zhang. Deployment of a Large-scale Peer-to-Peer Social Network. In *Proc. of WORLDS*, 2004.

[45] M. Yang, Z. Zhang, X. Li, and Y. Dai. An Empirical Study of Free-Riding Behavior in the Maze P2P File-Sharing System. In *Proc. of IPTPS*, 2005.

A Proof of Theorem 5.1

Consider a set $\mathcal{P}$ of P puzzles ($|\mathcal{P}| = P$), and let puzzle p, $1 \leq p \leq P$, be denoted by $\langle K_1^p, \hat{h}^p \rangle$. Define sets $I_\ell^p = \{f_{K_2^p}^2(1), \ldots, f_{K_2^p}^2(k)\}$ for $K_2^p = f_{K_1^p}^1(\ell)$; multisets $I^p = \bigcup_{\ell=1}^{L} I_\ell^p$; and multiset $I^{\mathcal{P}} = \bigcup_{p=1}^{P} I^p$. Recalling the definition of $\mathsf{Spread}(I^{\mathcal{P}}, s)$ from Section 5.1, we have:

Lemma A.1. *For any set $\mathcal{P}$ of P puzzles and any $\delta > 0$,*

$$\mathsf{P}\left[\neg\mathsf{Spread}\left(I^{\mathcal{P}}, (1+\delta)\frac{PkL}{n}\right)\right] \leq n\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{PkL/n}$$

Proof. Let $b^p_{\ell,i} = 1$ if $i \in I^p_\ell$ and $b^p_{\ell,i} = 0$ otherwise. So, the number of occurrences c_i of i in $I^{\mathcal{P}}$ is $c_i = \sum_{p=1}^{P}\sum_{\ell=1}^{L} b^p_{\ell,i}$. Then,

$$\mathsf{E}[c_i] \;=\; \sum_{p=1}^{P}\sum_{\ell=1}^{L}\mathsf{E}\left[b^p_{\ell,i}\right] \;=\; \sum_{p=1}^{P}\sum_{\ell=1}^{L}\mathsf{P}\left[b^p_{\ell,i}=1\right] \;=\; \sum_{p=1}^{P}\sum_{\ell=1}^{L}\frac{k}{n} \;=\; \frac{PkL}{n}$$

Now, using Chernoff bounds (e.g., see [33, Theorem 4.4]),

$$\mathsf{P}\left[c_i \geq (1+\delta)\mathsf{E}[c_i]\right] \;<\; \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mathsf{E}[c_i]} \;=\; \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{PkL/n}$$

and so

$$\begin{aligned}&\mathsf{P}\left[\neg\mathsf{Spread}\left(I^{\mathcal{P}}, (1+\delta)\mathsf{E}[c_i]\right)\right]\\ &\quad= \;\mathsf{P}\left[\bigvee_{i=1}^{n}\left(c_i \geq (1+\delta)\mathsf{E}[c_i]\right)\right] \;\leq\; \sum_{i=1}^{n}\mathsf{P}\left[c_i \geq (1+\delta)\mathsf{E}[c_i]\right] \;\leq\; n\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{PkL/n}\end{aligned}$$

□

The significance of Lemma A.1 is that it limits the number of index-sets for which a file bit obtained by an adversary can be useful. That is, any index the adversary queries of file is contained in only fewer than $(1+\delta)\frac{PkL}{n}$ index-sets, with probability specified in Lemma A.1.

Proof of Theorem 5.1. Index the adversaries by $1, \ldots, A$ and the puzzles by $1, \ldots, P$. Let S^{ap} be a binary random variable such that $S^{ap} = 1$ if adversary a produces the solution for puzzle p, and $S^{ap} = 0$ otherwise. Letting $S = \sum_{a=1}^{A}\sum_{p=1}^{P} S^{ap}$, we want to bound $\mathsf{E}[S]$ from above. For any $\delta > 0$ and $s = (1+\delta)\frac{PkL}{n}$,

$$\begin{aligned}\mathsf{E}[S] \;&=\; \mathsf{E}\left[S \mid \mathsf{Spread}(I^{\mathcal{P}}, s)\right]\mathsf{P}\left[\mathsf{Spread}(I^{\mathcal{P}}, s)\right] + \mathsf{E}\left[S \mid \neg\mathsf{Spread}(I^{\mathcal{P}}, s)\right]\mathsf{P}\left[\neg\mathsf{Spread}(I^{\mathcal{P}}, s)\right]\\ &\leq\; \mathsf{E}\left[S \mid \mathsf{Spread}(I^{\mathcal{P}}, s)\right] + Pn\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{PkL/n} \qquad (22)\end{aligned}$$

The second term is obtained by taking

$$\mathsf{E}\left[S \mid \neg\mathsf{Spread}(I^{\mathcal{P}}, s)\right] \leq P \quad \text{and} \quad \mathsf{P}\left[\neg\mathsf{Spread}(I^{\mathcal{P}}, s)\right] \leq n\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{PkL/n}$$

with the latter resulting from Lemma A.1.

Let q^{ap}_{hash} denote the number of queries of the form $\mathsf{hash}(K^p_1, *, *)$ made by adversary a, and let q^a_{file} denote the number of queries by adversary a to file. Under the constraints

$$\sum_{a=1}^{A} q^a_{\mathsf{file}} \leq A q_{\mathsf{file}} \qquad \sum_{p=1}^{P} q^{ap}_{\mathsf{hash}} \leq q_{\mathsf{hash}} \text{ for } 1 \leq a \leq A$$

we now seek to bound

$$\begin{aligned}
\mathsf{E}\left[S \mid \mathsf{Spread}(I^{\mathcal{P}}, s)\right] &= \sum_{a=1}^{A}\sum_{p=1}^{P} \mathsf{P}\left[S^{ap} = 1 \mid \mathsf{Spread}(I^{\mathcal{P}}, s)\right] \\
&\leq \sum_{a=1}^{A}\sum_{p=1}^{P} \frac{1}{L}\left(\frac{s q^a_{\mathsf{file}}}{k - \log_2(q^{ap}_{\mathsf{hash}} + L) - 1} + 1\right) \qquad (23) \\
&= \frac{1}{L}\left(\sum_{a=1}^{A} s q^a_{\mathsf{file}} \sum_{p=1}^{P} \frac{1}{k - \log_2(q^{ap}_{\mathsf{hash}} + L) - 1}\right) + \frac{AP}{L} \qquad (24)
\end{aligned}$$

where (23) follows from Theorem 5.2 and the fact that $\mathsf{Spread}(I^{\mathcal{P}}, s) \Rightarrow \mathsf{Spread}(I^p, s)$. In order to bound this, we first focus on

$$\sum_{p=1}^{P} \frac{1}{k - \log_2(q^{ap}_{\mathsf{hash}} + L) - 1} \qquad (25)$$

Using the technique of LaGrangian multipliers, define

$$\Lambda(q^{a1}_{\mathsf{hash}}, \ldots, q^{aP}_{\mathsf{hash}}, \lambda) = \left(\sum_{p=1}^{P} \frac{1}{k - \log_2(q^{ap}_{\mathsf{hash}} + L) - 1}\right) + \lambda\left(\sum_{p=1}^{P} q^{ap}_{\mathsf{hash}} - q_{\mathsf{hash}}\right)$$

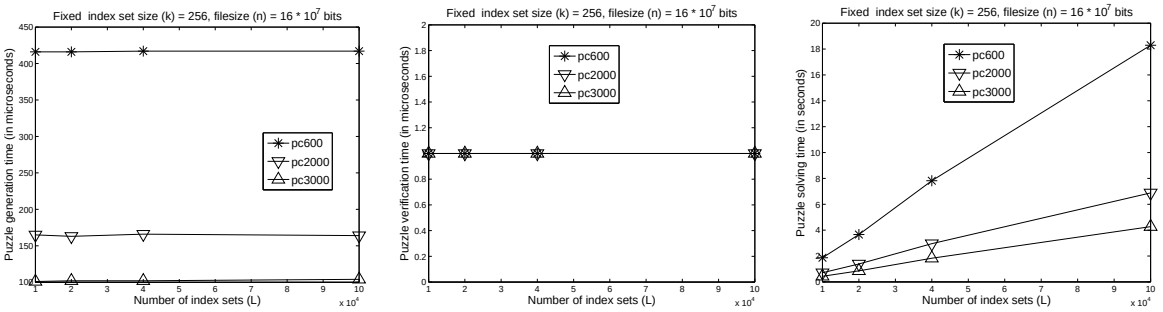
Since setting $\partial\Lambda/\partial q^{ap}_{\mathsf{hash}} = 0$ yields an identical constraint for each $1 \leq p \leq P$, the summation (25) is maximized when $q^{ap}_{\mathsf{hash}} = q^{ap'}_{\mathsf{hash}}$ for any $1 \leq p, p' \leq P$, i.e., $q^{ap}_{\mathsf{hash}} = q_{\mathsf{hash}}/P$ for each $1 \leq p \leq P$. So, the maximum value of (25) is $P/(k - \log_2(q_{\mathsf{hash}}/P + L) - 1)$ and the maximum value of (24) is

$$\frac{AP}{L}\left(\frac{s q_{\mathsf{file}}}{k - \log_2(q_{\mathsf{hash}}/P + L) - 1} + 1\right)$$

Plugging this into (22) gives the result. □

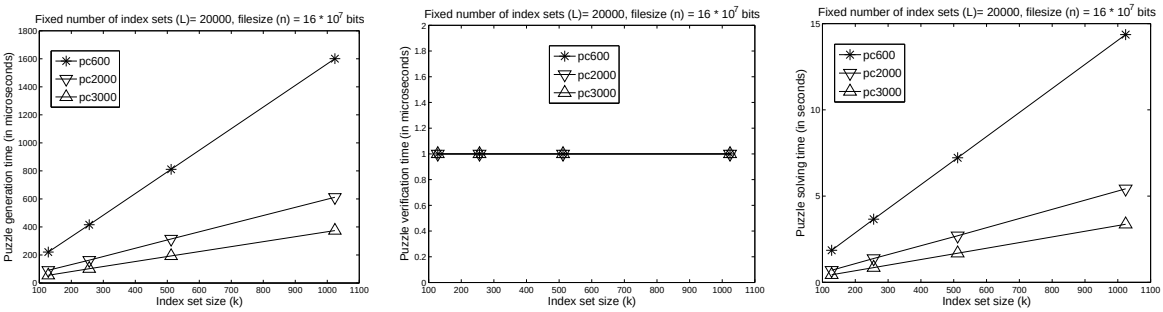
B Microbenchmarks

We implement the puzzle generation, verification, and puzzle solution algorithms in C++ using the OpenSSL libray SHA and AES implementations for hash and pseudorandom functions, respectively. We analyze performance on various node configurations: pc600 (600MHz Intel Pentium III "Coppermine" processor, 256MB PC100 ECC SDRAM), pc2000 (2.0 GHz Pentium IV processor,



(a) Generation (b) Verification (c) Solution

Figure 6: Microbenchmarks for puzzle operations on different CPU platforms as a function of L



(a) Generation (b) Verification (c) Solution

Figure 7: Microbenchmarks for puzzle operations on different CPU platforms as a function of k

512MB 400Mhz RAMBUS RAM), and pc3000 (3.0 GHz 64-bit Xeon processor, 2GB 400Mhz DDR2 RAM).

For each experiment, we generate 100 puzzles and report the average time taken to generate, verify, and solve the puzzles on the different hardware configurations as a function of L and k in Figures 6 and 7, respectively. First, note that the generation and verification times are orders of magnitude smaller than the puzzle solution time on the different platforms. Second, the generation and verification times are independent of L. These satisfy our original system requirements in Section 3: the verifier incurs low cost for generation and verification, and the verifier can adjust the difficulty of the puzzle by increasing L without incurring additional cost.