Making Contribution-Aware Peer-Assisted Content Distribution Robust to Collusion Using Bandwidth Puzzles

Michael K. Reiter[1], Vyas Sekar[2], Chad Spensky[1], Zhenghao Zhang[3]

May 28, 2009
CMU-CS-09-136[4]

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1]Department of Computer Science, University of North Carolina, Chapel Hill, NC, USA
[2]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA
[3]Computer Science Department, Florida State University, Tallahassee, FL, USA

[4]This report supersedes Computer Science Technical Report CMU-CS-08-156. It is an updated version of the report of September 2008.

Keywords: Peer-to-peer systems, Collusion, Shilling, Client puzzles

Abstract

Many peer-assisted content-distribution systems reward a peer based on 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., a group of colluding attackers can earn rewards by claiming to have served content to one another, when they have not. We propose a *bandwidth puzzle* mechanism to make contribution-aware peer-assisted content distribution robust to such collusion. Our construction 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 bound (in the random oracle model) the adversaries' ability to defeat our puzzle scheme. We also describe our integration of bandwidth puzzles into a fully operational media streaming system, and demonstrate the attack resilience offered by bandwidth puzzles through simulations of both streaming and file-sharing systems.

1 Introduction

Many systems that distribute content with the help of peer-to-peer (P2P) overlays measure peer contribution and incentivize participation. Peers who contribute more are rewarded with better performance via higher priority in the distribution overlay (e.g., [38, 32, 28]) or priority service through server-assisted downloads (e.g., [37]), or with other mechanisms (e.g., discount coupons [37]). We refer to such systems as *contribution-aware* peer-assisted content distribution systems.

Unfortunately, mechanisms for demonstrating how much data a peer has served are vulnerable to a simple form of "shilling" [12, 7], where colluding attackers report receiving service from each other without actually transferring content among themselves. In some systems, these attackers can degrade the system, e.g., by gaining a powerful position in the distribution overlay and then launching a denial-of-service attack [38, 32]. In others, this enables them to get higher priority 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., [28, 36, 29, 1]). 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.

We propose a *bandwidth puzzle* mechanism to make contribution-aware P2P content distribution robust to collusion attacks. With this mechanism, a *verifier* can confirm that claimed transfers of content actually occurred. For example, in P2P media streaming from a distinguished server (e.g., [38, 11, 32, 22]), or in P2P systems that have a distinguished node for tracking content-transfer transactions (e.g., [28, 37, 18]), this distinguished node can naturally play the role of the verifier.

There are two key insights behind our design. First, to those peers (or "provers") claiming to have content, the verifier presents puzzles for which the solution depends on the content. That is, the solution is computationally simple for a prover who has the content, but more difficult for a prover who does not. In this respect, our puzzle design is related to *proofs of data possession* (e.g., [3, 17]) and similar mechanisms; we detail the differences of our design in Section 2. Second, the verifier *simultaneously* presents these puzzles to all peers who currently claim to have the content, so as to make it difficult for a few peers who have the content 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 [13]; again, we detail the 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. The verifier can either deny or revoke credits granted in these transactions.

Our design is relatively simple, though its security analysis is more subtle than it 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 content bits that each possesses. We provide (in the random oracle model) a 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 content bits each possesses at the time the puzzles are issued and the numbers of hash computations and additional content bit retrievals that each adversary can perform in θ time. For example, this bound implies that for content of size n, an instance of our puzzle construction ensures that all adversaries claiming to have the content must download $\Omega(n)$ content bits to solve their puzzles in expectation, even if they retrieve up to n^ϵ bits on average before the puzzles are issued, for some constant $\epsilon < 1$. Moreover, this puzzle construction is efficient: It enables the verifier to construct each puzzle in $n \ln \frac{n}{n-n^\beta} + O(1)$ pseudorandom function computations in expectation and two hash function computations, for a configurable constant $0 < \beta < 1$, and to verify each puzzle in one comparison of hash function outputs. (Note that $\ln \frac{n}{n-n^\beta} = o(1)$, and so $n \ln \frac{n}{n-n^\beta} = o(n)$.) An honest prover invests $\frac{1}{2} n^{1+\alpha} \ln \frac{n}{n-n^\beta} + O(n^\alpha)$ time in expectation to solve this puzzle, for a configurable constant $\alpha > 0$ such that $\alpha + \beta > 1$.

We demonstrate the viability of bandwidth puzzles by integrating them into a functional multimedia streaming application. We demonstrate that a single verifier can scale to challenging thousands of peers simultaneously with puzzles, even while streaming content to other clients, and that puzzle distribution and solving introduce minimal jitter into the stream. We demonstrate the benefits of bandwidth puzzles against attacks in a simulated large-scale P2P streaming deployment, where we show that puzzles improve the legitimate clients' stream quality 40-300% (depending on the number of attackers) and reduce the attackers' quality by more than 2×. Moreover, the puzzle scheme limits the impact of such attacks by providing legitimate clients with performance nearly identical to the scenario when there are no attackers in the system. Finally, we describe simulations of a large-scale P2P file-sharing application in which bandwidth puzzles increased the legitimate client requests satisfied by 11-70%, and decreased the attackers' by 60-95%. Also, the mean download time of legitimate client requests decreased by 12-50%, and increased for attackers by 60-200%.

To summarize, the contributions of this paper are: (i) the design of bandwidth puzzles (Section 4), a practical defense against a documented form of attack on P2P systems; (ii) an analysis of our construction (in the random oracle model) that bounds the success attainable by adversaries against it (Section 5, Appendix A); (iii) implementation and evaluation of our construction in a functional peer-assisted streaming application (Section 6); and (iv) a demonstration of the benefits of puzzles on a simulated large-scale P2P deployments (Sections 8, 9).

2 Related Work

Incentives in P2P systems: Several works have demonstrated the limitations of P2P protocols in the presence of selfish or malicious users [16, 36]. Rewarding peer contributions has been suggested to overcome these limitations (e.g., [38, 16]), but these mechanisms cannot prevent colluding attackers from *freely* granting each other credits for fake transactions. *Bilateral* (tit-for-tat) mechanisms such as BitTorrent appear robust to collusion attacks. However, several studies (e.g, [16, 31, 26, 2]) have pointed out the limitations of bilateral mechanisms, and make the case for designing more *global* contribution-aware mechanisms.

By equating peers' debit and credit amounts for receiving and providing service, respectively, collusion can be made to yield no net gain (e.g., [37]). However, there are valid reasons to not equate the debit and credit amounts, such as asymmetries in upload and download bandwidth, and social considerations (e.g., [28]). Some global contribution-awareness schemes use pricing mechanisms (e.g., [5]), some of which are theoretically collusion-resistant (e.g., [2]). Currency management presents practical challenges for these schemes, however, requiring mechanisms to bootstrap new users in a Sybil-proof manner and to always ensure that there is appropriate currency in the system despite churn to achieve rapid price convergence and sufficient liquidity. Bandwidth puzzles are an alternative to implement collusion resistance that avoids currency management challenges, by seeking instead to directly detect when collusion (including with Sybils) occurs.

Failure to report transactions or solve puzzles: Clients are responsible for reporting transactions and solving puzzles in order to grant uploaders credits for the transaction. This raises the possibility of downloaders failing to report transactions or solving the puzzles and thus not giving adequate credit to their uploaders. This problem is orthogonal to the collusion attacks we consider and can be addressed by using fair-exchange [37] or proof-of-service [27] mechanisms.

Client puzzles: Client puzzles (e.g., [15, 23, 14]) force clients to demonstrate a certain proof-of-work to a server. This is used to throttle the number of requests that a client can issue 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 the content transferred and issuing puzzle challenges simultaneously.

Sybil attacks: Our adversary model – colluding attackers claiming to have contributed more resources than they actually have – is similar to a Sybil attack, which Douceur [13] suggests can be detected using simultaneous puzzle challenges. These puzzles validate that each claimed "identity" owns a certain amount of computation resources. Bandwidth puzzles instead validate that each client has expended a certain amount of communication resources.

Proofs of data possession (PDP) and retrievability (POR): Proofs of data possession (e.g., [3, 17, 4]) and proofs of retrievability (e.g., [24, 9, 35]) enable a user to verify that a remote store has not deleted or modified data the user had previously 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 colluding adversaries claiming credit for fake transactions. Second, PDP schemes minimize the communication between the prover and the verifier, without 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 scheme. In particular, 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 many provers.

One driving consideration in PDP/POR design is that the verifier no longer possesses the file about which it is querying. We present our bandwidth puzzle scheme in a framework in which the verifier possesses the content at the time it creates the bandwidth puzzle. However, in cases where this is undesirable, we can borrow an idea exploited in some PDPs to precompute puzzles and their solutions and either save them locally or outsource their storage, encrypted and authenticated, of course (e.g., [4]). When our approach is applied to a P2P file-sharing system, a natural realization of this idea would be to store the encrypted puzzles with the file itself. Once precomputed puzzles are exhausted, the file could be retrieved in full and more created. We emphasize, however, that many settings in which we are interested (e.g., multimedia streaming of live events, P2P CDNs) lend themselves to having a verifier with access to the content being transferred.

3 System Model and Goals

Our system model consists of a designated *verifier* and a collection of untrusted *peers*, also called *provers*. Any node can act as a verifier, provided that it can obtain the list of peers that purport to possess certain content and it has access to that content. P2P-assisted CDNs (e.g., [18], `www.pandonetworks.com/cdn-peering`), P2P assisted file-hosting (e.g., `www.vipeers.com`), and P2P streaming (e.g., [11, 38, 32]) have a central authority that can serve this role.

We are agnostic to how peers choose other peers for downloading. We require that peers report to the verifier the content they claim to have downloaded from others, and consequently the content that each has. Note that these requirements are already satisfied by many of the contribution-aware P2P systems we seek to protect (e.g., [38, 28]). The goal of our mechanism is to enable the verifier to ensure that the claimed bandwidth expenditures to transfer that content actually occurred.

The verifier does this by simultaneously presenting puzzles to the peers claiming to have certain content, 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 stable over the timescales involved in puzzle solving [42]. 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 peers suspected of not having the claimed content. The verifier can then take appropriate actions to ensure that the uploaders for these transfers do not receive credits for these transactions.

These puzzles 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, bandwidth puzzles must also ensure that for colluding provers to solve their puzzles within time θ, the content each receives in doing so, on average (possibly before receiving the puzzle itself), is of size roughly proportional to the full content size. 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 content, who could solve the puzzle and return it to the challenged prover. By (ii) above, the puzzle and 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 number of puzzles it can solve in time θ.

The above goal comes with two caveats. First, without network support, it is not possible for the verifier to ascertain which (if any) of the colluders actually has the content, even if it detects one or more of them as colluders via our scheme. For example, a prover with the content could invest its time in solving another prover's puzzle, at the expense of solving its own. As such, the verifier detects provers who collude, but cannot detect who has the content. Second, it is necessary that the content not be substantially compressible. If it were, then provers could exchange the compressed version in lieu of the original, and our goal could not be achieved. As such, in the rest of this paper we treat the content as random, 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 the 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: We use two hash functions: $\mathsf{hash} : \{0,1\}^\kappa \times \{1 \ldots L\} \times \{0,1\}^k \rightarrow \{0,1\}^\kappa$ and $\mathsf{ans} : \{0,1\}^k \rightarrow \{0,1\}^\kappa$. (Hash functions typically take a single string as input; we can encode the three inputs to hash in an unambiguous fashion as a single string input.) To prove security of our construction in Section 5, we model hash as a random oracle, though collision-resistance of ans suffices.[1]

Pseudorandom functions: A pseudorandom function family $\{f_K\}$ is a family of functions parameterized by a secret key $K \in \{0,1\}^\kappa$. Informally, 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 [19] for a formal definition. We use 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 $k \leq n$, where n is the content size in bits. We will discuss efficient implementations for f^2 below.

[1] Minimizing reliance on random oracles is desirable, since they are not a standard cryptographic assumption [10].

Construction: The puzzle verifier generates puzzles to challenge a collection of provers simultaneously. Generally, we assume that the verifier generates one puzzle per prover, though there is no obstacle to sending multiple puzzles per 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 content indices, i.e., uniformly random samples from $\{1 \ldots n\}$, without replacement. The verifier computes $\hat{h}$ as the hash of the content bits indexed by a randomly chosen index-set, appended together in an unambiguous order. Solving the puzzle means finding which of the L index-sets has this property and, more specifically, the string that hashes to $\hat{h}$. This requires at most L computations of hash for a prover who possesses the content, but could require substantially more for a prover who is missing some of the content indexed by the index-sets in the puzzle.

This construction, as described, would be inefficient. First, sending 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 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 is for the prover to return $\mathsf{ans}(str)$ for the string str satisfying $\hat{h} = \mathsf{hash}(K_1, \hat{\ell}, str)^2$, 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).

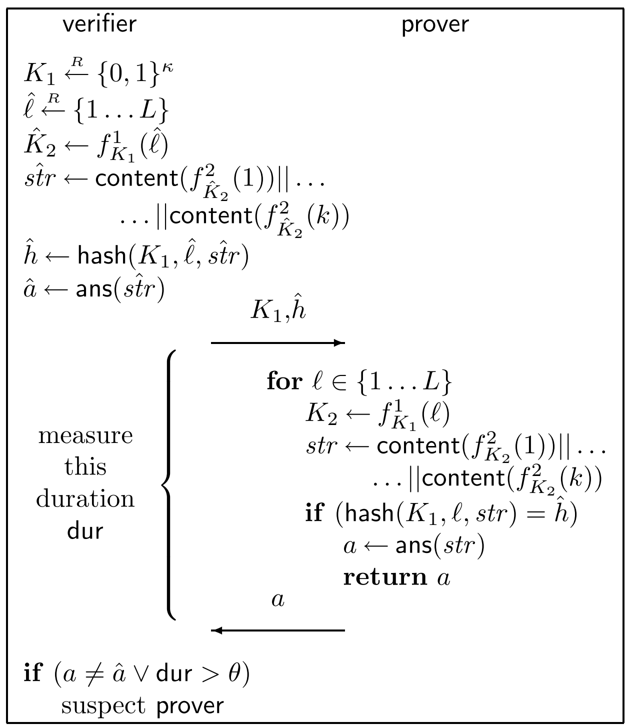


Figure 1: One bandwidth puzzle

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

Finally, a subtle but important implementation challenge arises for f^2, because our security analysis in Section 5 requires that f^2 be injective.[3] A natural approach to implement f^2, then, would be as a pseudorandom permutation (PRP) on $\{1 \ldots n\}$, but known constructions of PRPs for small domains from ones for larger domains (e.g., AES) are relatively quite expensive (e.g., [8]). The approach we use here exploits the fact that for any given key K, f^2_K is evaluated in our construction on all of $1 \ldots k$ anyway. Specifically, for a pseudorandom function family $\{f^3_K : \{1, 2, \ldots\} \to \{1 \ldots n\}\}$, we define $f^2_K(k')$ to be the k'-th *distinct* value in the sequence $f^3_K(1), f^3_K(2), \ldots$; i.e., we "skip over" repeat outputs from f^3_K. For this implementation, we prove the following in Appendix A:

Theorem 4.1 *The construction of Figure 1 has (i) expected puzzle generation cost of one* hash *computation, one* ans *computation, and* $n \ln \frac{n}{n-k} + O(1)$ *pseudorandom function computations, and (ii) expected puzzle solution cost (by an honest prover) of* $\frac{1}{2}L$ hash *computations, one* ans *computation, and* $\frac{1}{2}Ln \ln \frac{n}{n-k} + O(L)$ *pseudorandom function computations.*

When interpreting Theorem 4.1, it is important to note that $\ln \frac{n}{n-k} = o(1)$ for any $k = o(n)$, e.g., $k = n^\beta$ for $0 < \beta < 1$, as discussed in Section 5. So, the cost of puzzle generation is sublinear in n.

5 Security

For proving the security of our construction, first recall that we assume that $\{f^1_K\}$ and $\{f^2_K\}$ are pseudorandom function families [19], and that ans is a collision-resistant hash function. These primitives achieve their desired properties — 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, for the rest of this paper, we assume that these properties hold, ignoring events that occur with probability negligible in κ.

The hash primitive is modeled as a random oracle in our proof, which 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 content bits it receives before and during the puzzle-solving process. To model the receipt of content bits in our proof, it is also convenient to model a prover's retrieval of content bits as calls to a random oracle $\mathsf{content} : \{1 \ldots n\} \to \{0, 1\}$. As discussed in Section 3, our construction requires

[3] Relaxing this would introduce a term of $P^2 L \frac{\hat{k}(\hat{k}-1)}{2n}$ to the bound of Theorem 5.1 in Section 5, which is substantial for the values of n that we consider in Section 6.

[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 $\kappa \geq \kappa_0$.

that the content being exchanged have sufficient empirical entropy to be incompressible, as otherwise adversaries could "defeat" our verification by exchanging (in full) the compressed content. Thus, we model the content as a random string of length n, and track the number of bits that an adversary retrieves prior to returning a puzzle solution by the number of queries it makes to its $\mathsf{content}$ oracle.

In Appendix A, we present a proof of the following theorem, where $\Psi(x, m, p) = \mathsf{P}[X \geq x]$ for any binomially distributed random variable $X \sim \mathsf{B}(m, p)$.

Theorem 5.1 *Let* hash *and* $\mathsf{content}$ *be random oracles. Consider A collaborating adversaries, who are (i) collectively challenged to solve P puzzles; (ii) each permitted q_{hash} queries to* hash*; and (iii) collectively permitted Aq_{pre} queries to* $\mathsf{content}$ *before the distribution of the puzzles and Aq_{post} after. For any s and $\hat{k}$ satisfying $1 \leq s \leq PL$ and $\log_2(q_{\mathsf{hash}}+L)+2 \leq \hat{k} \leq k\left(1 - \frac{q_{\mathsf{pre}}}{n}\right) - 1$, the expected number of puzzles that these adversaries can solve collectively is at most*

$$\frac{AP}{L}\left(\frac{sq_{\mathsf{post}}}{\hat{k} - \log_2(q_{\mathsf{hash}} + L) - 1} + 1\right) + Pn\Psi\left(s, PL, \frac{k}{n}\right) + P^2L\Psi\left(k - \hat{k}, k, \frac{Aq_{\mathsf{pre}}}{n}\right) \qquad (1)$$

To see a consequence of Theorem 5.1, consider a constant number A of adversaries (i.e., constant as a function of n) challenged with a constant number P of puzzles (typically $P = A$) and that seek to each retrieve some $q_{\mathsf{pre}} \leq n^{\epsilon}$ content bits on average, where $0 \leq \epsilon < 1$, before the puzzles are issued. Suppose that $q_{\mathsf{hash}} = L$, and consider setting $L = n^{\alpha}$ for some $\alpha > 0$ and $k = n^{\beta}$ for some $0 < \beta < 1$ where $\alpha + \beta > 1$. Consider setting $\hat{k} = k - k(\delta + \frac{Aq_{\mathsf{pre}}}{n})$ for any constant $0 < \delta < 1$, in which case $\log_2(q_{\mathsf{hash}} + L) + 2 \leq \hat{k} \leq k\left(1 - \frac{q_{\mathsf{pre}}}{n}\right) - 1$ for sufficiently large n and we can show (using a Hoeffding bound[5]) that $P^2L\Psi\left(k - \hat{k}, k, \frac{Aq_{\mathsf{pre}}}{n}\right) \to 0$ as $n \to \infty$. Setting $s = (1+\delta')\frac{PLk}{n}$ for $\delta' > 0$ (and using a Chernoff bound[6]) implies $Pn\Psi\left(s, PL, \frac{k}{n}\right) \to 0$ as $n \to \infty$. For this value of s, Theorem 5.1 implies that $q_{\mathsf{post}} = \Omega(n)$ in order for the adversaries to solve P (or any constant number of) puzzles in expectation. This, in our opinion, is a strong result: to solve the P puzzles in expectation, each adversary must retrieve, on average, an amount of the content roughly proportional to its size, even if each retrieves, on average, up to n^{ϵ} bits of the content before the puzzles are issued.

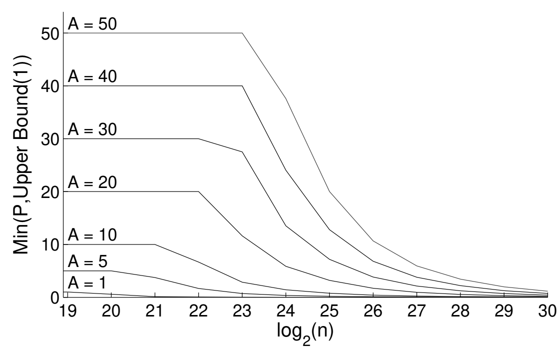


Figure 2: An example of Theorem 5.1

[5] $\mathsf{P}[X \geq \mathsf{E}[X] + \delta m] \leq e^{-2\delta^2 m}$ for $X \sim \mathsf{B}(m, p)$ [21].

[6] $\mathsf{P}[X \geq (1+\delta')\mathsf{E}[X]] \leq \left(e^{\delta'}/(1+\delta')^{(1+\delta')}\right)^{\mathsf{E}[X]}$ for $X \sim \mathsf{B}(m, p)$ [30, Theorem 4.4].

Examples of Theorem 5.1 for values of A and n are shown in Figure 2, which plots the minimum of P and the bound (1) for $P = A$, $L = \frac{1}{12}n^{71/100}$, $k = \frac{1}{4}n^{3/10}$, $q_{\mathsf{pre}} = n^{3/10}$, $q_{\mathsf{post}} = n^{3/10}$, $s = 21An^{1/100}$, and $\hat{k}$ chosen optimally in the range $\log_2(q_{\mathsf{hash}} + L) + 2 \leq \hat{k} \leq k\left(1 - \frac{q_{\mathsf{pre}}}{n}\right) - 1$. For these parameters, presenting puzzles every $n = 2^{22}$ bits $\approx$ 520KB suffices to detect half of five collaborating adversaries in expectation, and presenting puzzles for each $n = 2^{25}$ bits $\approx$ 4MB suffices to detect half of 50 collaborating adversaries in expectation. Moreover, our bound is loose in several respects, and so the detection capability of this approach is even better than shown in Figure 2.

6 Evaluation in a Media Streaming System

We implemented and evaluated a contribution-aware peer-assisted content distribution system augmented with bandwidth puzzles. The system is designed for streaming real-time media, e.g., a live broadcast of an event (c.f., [11, 38, 32]). It uses a real-time transport protocol (RTP [34], `jlibrtp.org`) to stream media to a set of *seed* clients; these clients can then stream this to other clients over a P2P overlay. The server also acts as the verifier. In this role, it maintains a persistent TCP connection with each client (including the seeds) over which puzzle challenges and responses are communicated for each n bits of the media stream. Each client solves puzzles using a separate thread from that which handles the stream. Our puzzle implementation uses AES to implement f^1 and f^3 (and hence f^2), and SHA-256 to implement hash and ans.

We evaluate our system on Emulab [40] using five classes of machines: 600MHz Pentium III with 256MB of memory (Class A); 850MHz Pentium III with 256MB of memory (Class B); 2GHz Pentium 4 with 512MB of memory (Class C); 3GHz 64-bit Xeon with 2GB of memory (Class D); and 2.4GHz Pentium Core 2 Duo with 2GB of memory (Class E). The server/verifier was a Class E machine. The server sends a 768Kbps stream[7] to 50 seed clients[8] over a 100Mb/s network. In addition, we configured the network with wide-area parameters in certain tests, as described below. In all our experiments, we fixed $L = \frac{1}{12}n^{71/100}$ and $k = \frac{1}{4}n^{3/10}$, and so the security bounds in Figure 2 are representative for our experiments.

We address the following questions:

- Can we set a suitable θ to accommodate heterogeneity in client capabilities?
- Does the puzzle solving overhead impact the user's quality of service?
- How many simultaneous puzzle challenges can the server handle?
- How do wide-area effects (e.g., latency, loss) affect the choice of θ?

Client heterogeneity and choice of n: We first examine the impact of n on puzzle-solving time and specifically the advantage that faster computers have over slower ones, since the threshold θ must allow for slower computers to reliably solve their puzzles. Figure 3 shows

[7]For example, ESPN360 requires 400Kbps and recommends 768Kbps, see `espn.go.com/broadband/espn360/faq#21`.

[8]As a point of comparison, the server in the popular P2P streaming system PPLive supports 25 seed clients at 400Kbps [22].

the ratio of the 95th percentile time for a Class-X machine (X $\in$ {A, B, C, D, E}) to the 50th percentile time for a Class-E machine. If the slowest clients that the server accommodates are of Class X, and the fastest are of Class E, then Figure 3 shows the number of puzzles that the Class-E client can solve in θ time, if θ is set so that the Class-X client can solve one puzzle reliably.

Figure 3 shows a large gap in puzzle-solving ability between the slowest and fastest machines. That said, the slowest machines would presumably not meet the minimum system requirements for viewing a live stream anyway; e.g., of the classes we consider, only D and E meet ESPN360's minimum requirements (see `espn.go.com/broadband/espn360/faq#21`). So, we discard Classes A and B (and conservatively include Class C) for the rest of our evaluation. Figure 3 then shows that an attacker with a Class-E machine can successfully impersonate roughly seven Class-C machines, and so could inflate his claimed transfers by 7×. While not ideal, this provides a limit on the extent to which an adversary can game the system. With this choice made, we further narrow our attention to puzzles for each $n = 2^{23}$ bits for the rest of our evaluation.

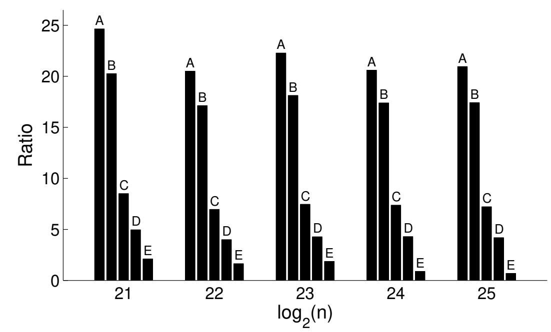


Figure 3: Ratio of 95th percentile puzzle-solving time for Class-X machine (X $\in$ {A, B, C, D, E}) to 50th percentile puzzle-solving time for Class-E machine during live streaming experiments.

Application impact: We now consider the impact on jitter of introducing puzzle solving into media streaming. Jitter [34] is an estimate of the statistical variance of the RTP (application layer) data packet interarrival time. Figure 4 shows the distribution of jitter of the media stream at clients for a duration including 100 puzzle challenges, for different machine classes. Figure 4 is a box-and-whiskers plot; each box shows the 25th percentile, median and 75th percentile values, and the whiskers extend to the 1st and 99th percentile values. As this figure shows, puzzles have little impact on jitter for any of Classes C–E.

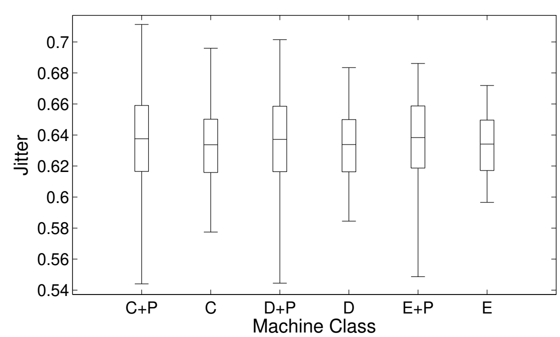
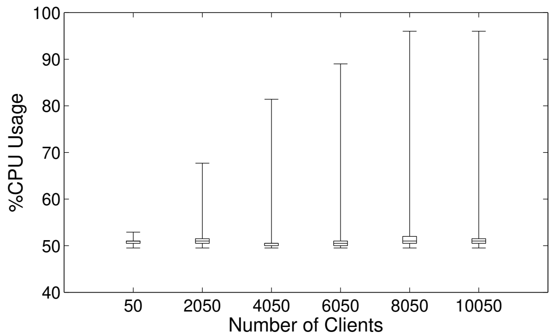
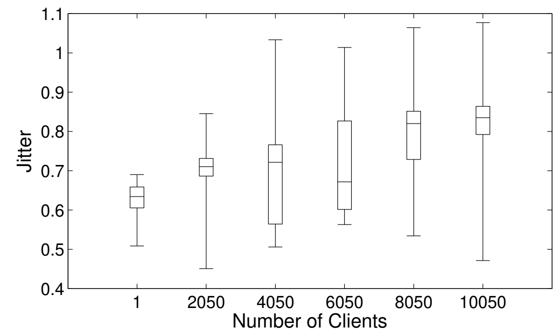


Figure 4: Jitter per machine class. "+P" indicates with puzzles.

Verifier scalability: To test scalability, we fixed the number of clients to which a Class E server streams content at 50, but had it simultaneously generate and send puzzles to a number of clients (in addition to these 50) ranging from 0 to 10000. Due to limits on the



(a) CPU usage for Class-E verifier node



(b) Jitter for one Class-E client

Figure 5: Scalability tests in which 50 clients receive stream from verifier, and a variable number of clients receive puzzle challenges from verifier.

number of available Emulab computers, we co-located the puzzle-receiving clients on a few machines, but still established an independent TCP connection to each one. We sampled the CPU and memory usage of the verifier (both user and system) during the tests at half-second intervals using `top`. Figure 5(a) shows the distribution of CPU usage for the verifier in such a test. The verifier's median and even 75th percentile utilization is largely unchanged by challenging 10050 clients, and also sending the stream to 50 of them. The 99th percentile does increase, though it never reaches 100%. (Memory utilization exhibited moderate growth, and far less variance. It topped out at less than 75% in the 10050-client case.) We also confirmed the simultaneity of puzzle distribution in these tests: the time between sending the first puzzle and receiving an application-level acknowledgement from the last client to which a puzzle was sent (i.e., the 10050th) was at most 450ms. It is clear that even a moderately well-provisioned verifier machine should scale beyond 10000 clients, and a machine with more cores and memory should easily scale far beyond that.

We also monitored one of the 50 clients receiving the media stream during these tests, to see the impact on its jitter as the number of puzzle-solving clients is increased. Figure 5(b) shows that the median jitter at this client when the server challenges 10050 (including this one) is within 50% of the median jitter when this client is served in isolation. This suggests that increasing the number of puzzle-solving clients has little impact on individual clients' stream quality.

Wide-area effects: The primary concerns with streaming in a wide-area setting are latency and packet loss. Uniformly increased latency simply means that the verifier waits correspondingly longer to receive puzzle solutions. If there is significant diversity across provers in the latencies to reach them, the verifier can send puzzles to more distant provers first, to increase simultaneity of distribution. (Geolocation by IP address can provide latency estimates that would be difficult for a prover to mislead.) Also, more puzzles or more difficult puzzles (i.e., by increasing n or L) can be used to minimize the effects of both latency variance across provers and transient latency variations per prover.

The more significant impact of wide-area streaming is the risk of increased packet loss. Distribution of puzzles over TCP helps to deliver puzzles and their solutions reliably, but the UDP-based RTP stream does not guarantee reliable delivery of stream packets. Consequently, during periods of high packet loss, an honest prover might be missing some of the content bits indexed in an index-set; if so, it searches through all possibilities for them. The effect of this searching on puzzle-solving time is shown in Figure 6, where the network packet loss rate ranges from 0% to 4%. Even 2% represents an unusual packet loss rate that, e.g., justifies a "warning" indication at a real-time monitoring site like `www.internetpulse.net`; a 4% packet loss rate is "critical". This figure shows that even at 2% loss, nearly 75% of the puzzle-solving times experienced are within those observed with 0% loss, and the 99th percentile is within twice those observed with 0% loss. So, doubling θ during periods of 2% loss (as indicated at, e.g., `www.internetpulse.net`) should allow adequate puzzle-solving time, or θ could be permanently doubled with the cost of allowing adversaries a more slack with which to gain slightly more advantage.

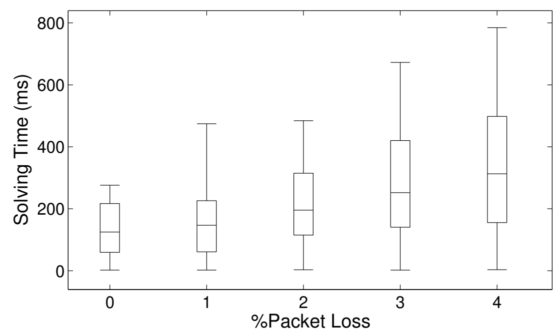


Figure 6: Puzzle-solving time for a Class-E client as a function of packet loss.

7 Simulation Framework

We demonstrate the benefits of using bandwidth puzzles in P2P *streaming* systems (Section 8) and in *file-sharing* applications representative of P2P-assisted file-hosting and CDN systems (Section 9). To this end, we implement an event-driven simulation framework to model such systems. There are three types of events in the simulation model: content exchanges, transaction reports, and puzzle challenges.

Incentive mechanism: We use an incentive scheme similar to Maze [28]. Content is transferred in *chunks* of 1MB in size, and we assume the existence of a central server to which peers authenticate and periodically report transactions on a per-chunk basis. This server maintains a per-peer "points system". Each peer earns 1.5 points for every chunk uploaded and consumes 1 point per chunk downloaded. In a system augmented with bandwidth puzzles, this server additionally plays the role of the verifier to issue and check puzzle challenges. New peers are given initial points to allow some free downloads before they can contribute. Each peer queues incoming requests (i.e., asking it to upload some content) in increasing order of $rqsttime - 3\log\rho$, where *rqsttime* is the request arrival time and ρ is the current number of points the requester has. Intuitively, requests that arrived earlier and requests from peers with more points are served earlier. We can configure the system to either allow lower priority service to free-riders (clients with zero points) or simply deny their requests.

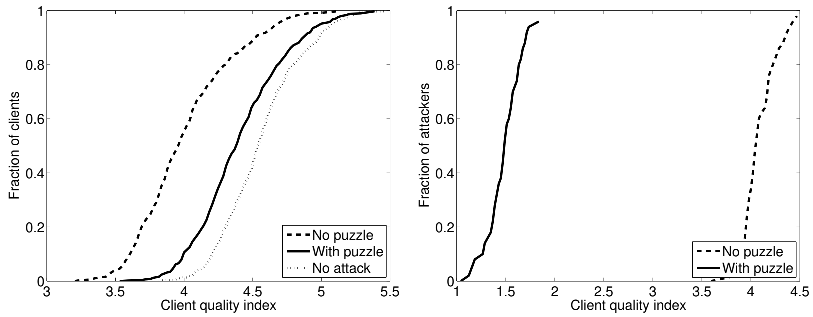
Adding bandwidth puzzles: In a traditional contribution-aware P2P system, on receiving a report of a completed transaction involving a specific content chunk, the server subtracts points from the downloader and credits points to the uploader. With a system augmented with bandwidth puzzles, handling transactions 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 specifying the identifiers of the uploader, downloader, and the content of which this chunk is a part (e.g., a filename or a segment of a video stream).

The server sends puzzle challenges in the role of the verifier. These puzzles are issued at a logical content granularity appropriate to the application (file-sharing or streaming) instead of at chunk granularity. 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 content for which an exchange was reported during the previous puzzle epoch, the server retrieves the set of all peers that currently claim to have this content and generates puzzles for these clients. The server issues puzzles for this content to these clients simultaneously. (The server can also issue puzzles for other content that a disjoint set of clients claim to possess. The requirement of disjointedness is to ensure that no client is tasked with solving two puzzles at the same time.) Upon receiving a response for a puzzle sent to a peer x for content $\mathsf{content}$, the server verifies if the answer is correct and if the response came within the puzzle-specific timeout θ. If so, the server *validates* the pending transactions pertaining to $\mathsf{content}$ in which x was the downloader and credits the uploaders of these pending transactions for transferring the content's chunk(s).[9]

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, where 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 content, even though y does not actually spend any bandwidth for the transfer. Each such x periodically reports fake transactions to the server in addition to its actual (legitimate) transactions, if any.

The notion of a collusion graph is general and can capture different collusion patterns. For example, in the Maze measurement study [28], the authors find that most collusion patterns comprise two or three mutually colluding nodes. This is represented as a directed *clique*. In our notation, for example, Clique(100,10) denotes that there are 100 attackers organized in cliques each of size 10 attackers that collude with one another. Other forms of observed collusion patterns include *star* topologies, each comprised of a real attacker and a collection of Sybil identities for that attacker. The only role of the Sybil identities is to grant points to their master node and they do not generate any real transfer requests. In our notation, for example, a Star(200,19) graph denotes that there are 200 nodes in the graph organized in 10 star graphs, each having 19 leaf nodes representing the fake (Sybil) identities and the actual attacker as the "center" of the star. We focus on collusions of moderate size since the Maze measurement study [28] found that most collusion patterns involved a small

[9]Detecting downloaders that habitually refuse to solve puzzles is a separate problem that can be solved using fair-exchange or proof-of-service mechanisms; see Section 2.



(a) Legitimate clients

(b) Attackers

Figure 7: Benefits in a P2P streaming system

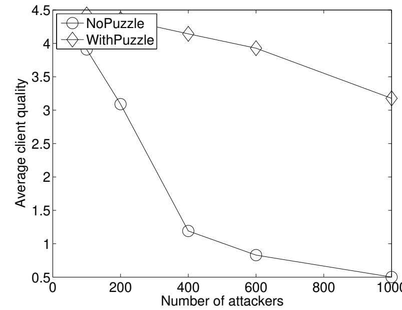


Figure 8: Varying the number of attackers

number of attackers.

Simplifying assumptions: We make some simplifying assumptions to make our simulation framework scalable. 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 [6]. Also, we assume that all content is split into 1MB chunks, and that all content exchanges and transaction reports happen at the granularity of this chunk size. To simplify the request queue dynamics, each peer queues content transfer requests based on the downloader's points using the prioritization mechanism described above and serves these requests one at a time, without preemption.

8 Benefits in P2P Streaming Systems

In this section, we show via simulation the benefits of using bandwidth puzzles in a contribution-aware peer-assisted streaming application (e.g., [11, 38, 32]).

System description: We assume that the multimedia stream is divided into discrete epochs of size 1000 units of simulation time where each unit corresponds to 100ms of real time. The content within each epoch is divided into suitably encoded *stripes* [11, 38]. This encoding (e.g., [20]) has the property that a client can access the original content as long as it is able to download *at least* one stripe and it receives better performance (e.g., higher video quality) if it downloads more stripes. As discussed in Section 7, each stripe is broken into 1MB chunks and peers download the chunks corresponding to each stripe using a suitable lookup mechanism.

In the context of the simulation framework discussed in Section 7, the puzzle epoch coincides naturally with the streaming epoch and the puzzle challenges happen at a *per stripe* granularity. In this setup, each client caches the stripes it received in the previous epoch to answer puzzles. The storage overhead for caching the stripes is quite low — only 2MB per stripe in our simulations.

Simulation parameters: Each streaming session lasts 50 epochs with all clients and attackers arriving at the start of the session. We assume that there are 10 stripes, each of size 2MB. In each epoch, the server bootstraps 5 seed nodes in the system with the 10 stripes for the next epoch. Some clients initially download stripes from these seed nodes and subsequently serve these to others. For each result, we repeat the simulation five times and present the averages across the multiple runs.

We consider a scenario where attackers create fake identities that pretend to receive the stream. This helps attackers download more stripes (higher stream quality) and receive content directly from the seed nodes (higher priority service). We assume that a puzzle sent to a peer who does not have the stripe for which the puzzle was generated (or a fake peer) is solved with a fixed probability 0.1. In the Star(200,19) case, for example, this means that if the 19 Sybil identities in one star each report one stripe transfer from the node at the center, then in expectation 19 stripes $\times\, 2\, \frac{\text{chunks}}{\text{stripe}} \times 0.1 = 3.8$ of these fake chunk transfers get validated.[10]

Performance benefits: We define the *user quality* to be the average number of stripes received by a client per epoch in the streaming session. Figure 7(a) shows the CDF of the client quality in a streaming system with 100 legitimate clients under three scenarios: no attack, under a Star(200,19) attack without the puzzle scheme, and under a Star(200,19) attack with the puzzle scheme in place. We see that when the puzzle scheme is used the client quality with an attack is very close to a system without attackers. In Figure 7(b), there is more than $2\times$ reduction in the median attacker quality when bandwidth puzzles are used. Figure 8 shows the average legitimate client quality as a function of the attack size. Each attack is of the form Star(X,19) where X is the number of attackers. As the number of attackers grows, the decrease in quality is less severe when the puzzle scheme is used. These results confirm that bandwidth puzzles can improve legitimate client performance and deter attackers in P2P streaming systems.

9 Benefits in File-sharing and P2P-CDN Systems

There are several P2P file-sharing applications that can incorporate and benefit from bandwidth puzzles: peer-assisted CDNs (e.g., [18, 25], `www.pandonetworks.com/cdn-peering`), and peer-assisted file-hosting services (e.g., [41], `www.vipeers.com`). These satisfy the requirement for a central distinguished node to serve the role of a verifier, having access to the content being shared, a list of active peers, and the content they host.

System description: As representative evaluations, we show the benefits in *file-sharing* (representing file-hosting and CDN services) in the context of a Maze-like [41] P2P system. Our choice of Maze was motivated by two factors. First, Maze 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, Maze incentivizes peers to upload content based on the points system described in Section 7. Such incentive mechanisms are important for the viability of P2P systems to encourage uploaders. However, a subsequent measurement study

[10] Since 190 identities are fake, the attackers' resources correspond to $A = 10$. Because the verifier issues puzzles per stripe ($\log_2(n) \approx 24$), the value of (1) for $A = 10$, $P = 200$, and, e.g., $L = \frac{5}{3}n^{71/100}$ and $q_{\mathsf{post}} = n^{1/10}$ (and otherwise the same parameters used for Figure 2) is consistent with this choice.

demonstrated a wide range of collusion attacks [28]. Our goal is to minimize the impact of collusion in such systems using bandwidth puzzles.

In the context of our simulation framework (Section 7), puzzle challenges occur periodically over a suitable puzzle epoch size at a *per file* granularity.

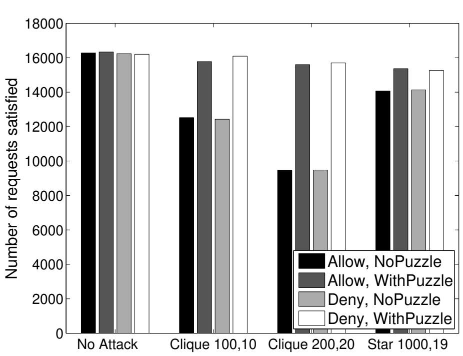
Simulation parameters: We evaluate two configurations: one in which peers allow free-riders (but give their requests very low priority), and another in which peers deny service to free-riders. Each of our simulations runs for 10^5 units of simulation time where each unit of simulation time corresponds to 100ms of real time. There are 100 files shared with file request popularity following a Zipf distribution. Each file is an integral number of chunks in size, chosen uniformly at random from the range $[4, 10]$. 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. Unless stated otherwise, we assume that a puzzle sent to a peer who does not have the file for which the puzzle was generated (or a fake peer) is solved with a fixed probability 0.1. For each result, we repeat the simulation five times and present the averages across the multiple runs.

Performance benefits: We focus on two metrics in our simulations: number of file 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 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.

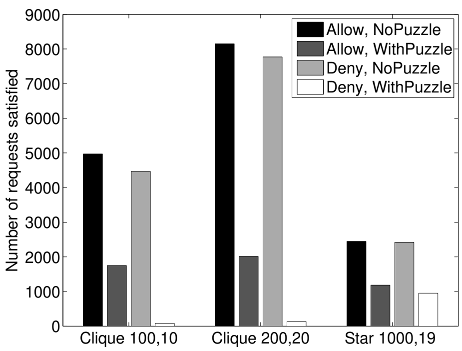
Figure 9(a) shows the number of legitimate clients' requests satisfied. Bandwidth puzzles boost the performance by 11-70%. The benefit is slightly better when free-rider requests are not serviced. Also, across Clique(100,10) and Clique(200,20), as the number of attackers increases, the benefit provided by the puzzle mechanism increases two-fold. In Figure 9(b) we see that bandwidth puzzles decrease the total number of attackers' requests satisfied by 50-75% when free-riders are allowed, and by 60-95% when free-rider requests are denied.[11]

Figures 10(a) and 10(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 10(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 9(b) and the Maze system design. Recall that in Maze, each peer is given some initial credits that it can use to download files.

[11]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.

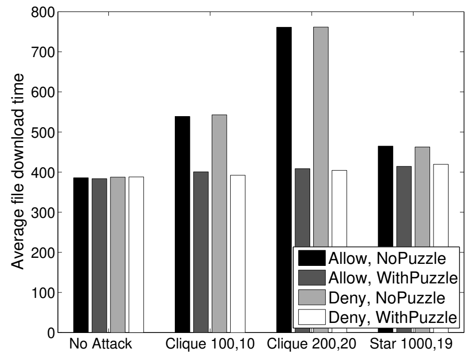


(a) Legitimate clients

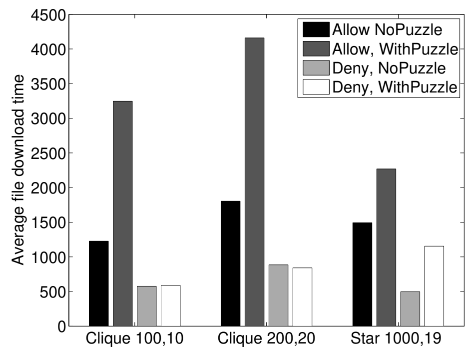


(b) Attackers

Figure 9: Number of requests satisfied



(a) Legitimate clients



(b) Attackers

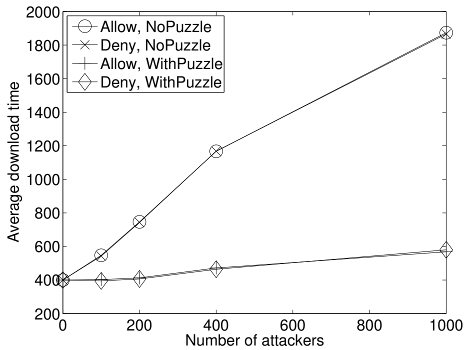
Figure 10: Mean file download time

Results for file-sharing workload tests. Each bar represents one of four 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.

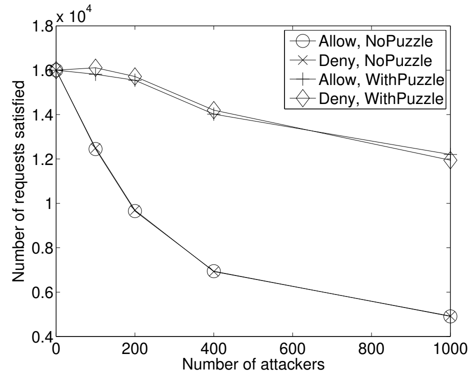
In the deny free-rider configuration (with bandwidth puzzles), the only attacker requests satisfied correspond to these initial *free* downloads. Since attackers have credits initially, these requests see smaller queueing delays.

Figures 9(a) and 10(a) also show the results when there are no attackers. 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.

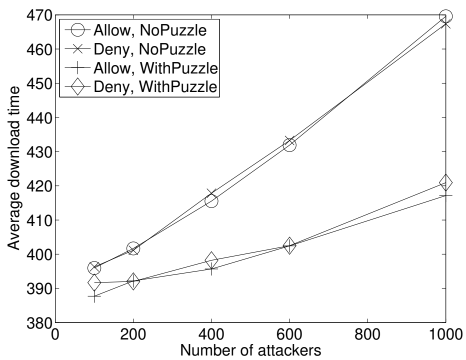
Sensitivity to attack parameters: In additional simulations, we varied the number of clique and star attackers to evaluate the impact on legitimate clients. For the clique attack, we set each clique size to 10 and for the star attack we set the star size to 19; i.e., for attack size = X, the attack is Clique(X,10) and Star(X,19). Figure 11 shows the average download time for legitimate clients as a function of the attack size. We observe that without bandwidth puzzles, legitimate clients see a more drastic drop in performance and the average



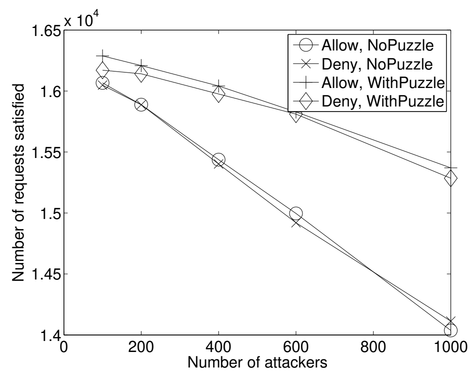
(a) Clique



(a) Clique



(b) Star



(b) Star

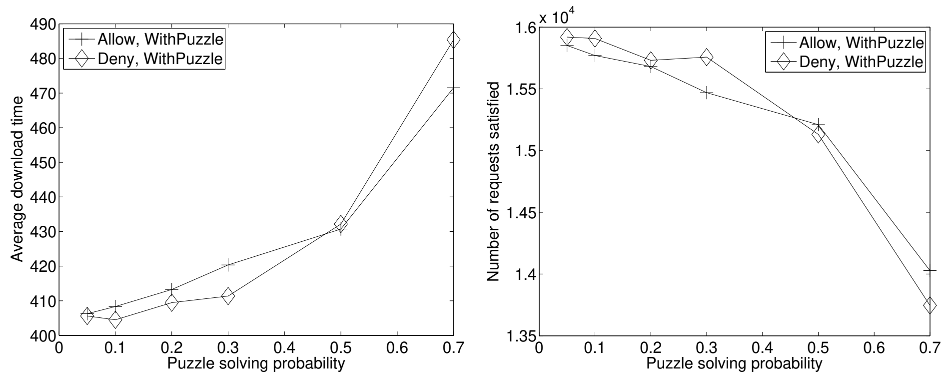
Figure 11: Average download time

Figure 12: Number of requests served

Varying the number of attackers in the file-sharing workload.

download time increases quite significantly. With puzzles, however, larger attacks have lesser impact. Similarly, in Figure 12, the number of legitimate client requests satisfied degrades more gracefully when bandwidth puzzles are used.

We also considered variations of the puzzle-solving probability for attackers. The effective puzzle solving probability is ultimately determined by a variety of factors: e.g., the computation power of the adversaries, how effectively the adversaries are able to divide the puzzle solving responsibilities among themselves, and the ability to utilize the natural churn in a file-sharing system to use "hidden" solvers as discussed at the end of this section. In order to understand the effect of increasing the attackers' puzzle solving capabilities, in Figure 13 we fix the attack to be Clique(200,10) and examine the impact of increasing the puzzle solving probability on the average download time and number of requests served to legitimate clients. The result shows that as long as the per-puzzle probability is less than 0.5, the impact of the attack on a system with puzzles is quite low. This is encouraging since it means that even if determined attackers were able to utilize these external factors, as long as their effective solving probability is within reasonable bounds, they cannot impact the



(a) Average download time (b) Number of requests served

Figure 13: Varying the puzzle solving probability in the file-sharing application with a Clique(200,10) attack

performance of legitimate clients.

Verifier scaling: The verifier maintains state for client contributions, pending transactions, and outstanding puzzles. Assuming a large workload of one million clients, pending transactions, and outstanding puzzles, the memory overhead per type of state was 28, 36, and 72 MB, respectively. Even if we assume that the server maintains these data structures in memory, these are modest requirements for commodity desktops today. A million transactions or puzzles in a single epoch (say 100 seconds) is large by the standards of many current P2P systems. For example, the Maze measurement study [41] showed that there were roughly 10000 simultaneous users, 3000 transactions, and 200K registered users. A similar measurement study on Gnutella [33] reported 40000 active nodes in a given snapshot.

Discussion: Adversaries in a static-file sharing system (e.g., versus a streaming system for live events) can use client churn to their advantage: they share a file among themselves, "leave" the system, and then solve puzzles to validate other claimed transfers from one collaborator remaining in the system to new, Sybil identities that they periodically insert into the system. Our puzzles limit the transfers for which the adversaries can gain credit to roughly the number of such "hidden" collaborators per puzzle-issuing period. Note that this attack is ineffective in the live multimedia streaming case: the content initially shared among the "hidden" adversaries becomes obsolete, both for viewing and gaining transfer credits.

10 Conclusions

Peer-assisted content distribution systems continue to be subject to adversaries exploiting weaknesses in the underlying incentive mechanisms. In particular, a group of colluding adversaries can implement a "shilling" attack, i.e., by reporting service from one another without spending any actual resources, to get preferential service. Our work provides a simple, yet powerful primitive to thwart such collusion attacks in peer-assisted content distribution systems. It is based on simultaneously challenging peers with *bandwidth puzzles* to demonstrate that the purported data transfers actually took place. We quantified the security of our scheme in the random oracle model. We also showed via an implementation in a functional streaming system that our puzzles cost little in terms of scalability or perceived stream quality. Finally, we showed by simulations that bandwidth puzzles prevent colluding attackers from gaining undue advantage via shilling attacks and from impacting the performance of honest peers.

Acknowledgements

We are grateful to Katie Benedetto for useful discussions, and to anonymous conference reviewers for various comments. This work was supported in part by NSF awards CNS-0326472, CT-0756998, and ANI-0331653.

References

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

[2] C. Aperjis, M. J. Freedman, and R. Johari. Peer-Assisted Content Distribution with Prices. In *Proc. CoNeXT*, 2008.

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

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

[5] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. Making P2P accountable without losing privacy. In *Proc. ACM WPES*, 2007.

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

[7] R. Bhattacharjee and A. Goel. Avoiding ballot stuffing in eBay-like reputation systems. In *Proc. ACM SIGCOMM P2P-ECON*, 2005.

[8] J. Black and P. Rogaway. Ciphers with arbitrary finite domains. In *CT-RSA '02*, pages 114–130, 2002.

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

[10] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. ACM STOC*, 1998.

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

[12] C. Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proc. ACM EC*, 2000.

[13] J. Douceur. The Sybil attack. In *Proc. IPTPS*, 2002.

[14] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Proc. CRYPTO*, 2003.

[15] C. Dwork and M. Naor. Pricing via processing, or, combatting junk mail. In *Proc. CRYPTO*, 1993.

[16] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *Proc. ACM EC*, 2004.

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

[18] M. J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with Coral. In *Proc. NSDI*, 2004.

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

[20] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, pages 74–93, Sept. 2001.

[21] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Stat. Assoc.*, 58(301):13–30, 1963.

[22] G. Huang. Keynote: Experiences with PPLive. In *Proc. ACM SIGCOMM P2P-TV Workshop*, 2007.

[23] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proc. NDSS*, 1999.

[24] A. Juels and B. S. Kaliski, Jr. PORs: Proofs of retrievability for large files. In *Proc. ACM CCS*, 2007.

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

[26] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In *Proc. P2P Econ*, 2004.

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

[28] 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 *Proc. ICDCS*, 2007.

[29] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting BitTorrent for fun (but not profit). In *Proc. IPTPS*, 2006.

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

[31] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. One hop reputations for peer to peer file sharing workloads. In *Proc. NSDI*, 2008.

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

[33] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. In *Proc. IEEE Internet Computing*, 2002.

[34] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications. IETF RFC 3550, July 2003.

[35] H. Shacham and B. Waters. Compact Proofs of Retrievability. `http://eprint.iacr.org/2008/073.pdf`, 2008.

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

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

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

[39] W. Uhlmann. Vergleich der hypergeometrischen mit der binomial-verteilung. *Metrika*, 10(1):145–158, 1966.

[40] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. OSDI*, 2002.

[41] M. Yang, H. Chen, B. Y. Zhao, Y. Dai, and Z. Zhang. Deployment of a large-scale peer-to-peer social network. In *Proc. WORLDS*, 2004.

[42] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker. On the Constancy of Internet Path Properties. In *Proc. IMW*, 2001.

A Proofs

Proof of Theorem 4.1. The result is by a "coupon collector" analysis. When generating $f^2_{K_2}(1)\dots f^2_{K_2}(k)$ for the ℓ-th index-set (i.e., $K_2 \leftarrow f^1_{K_1}(\ell)$), let X_i be a random variable denoting the number of computations of $f^3_{K_2}$ while having collected exactly $i-1$ *distinct* outputs of $f^3_{K_2}$. Then, X_i is geometrically distributed with parameter $p_i = 1 - \frac{i-1}{n}$, and $\mathsf{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}$. So, the expected number of computations of $f^3_{K_2}$ is $\mathsf{E}\left[\sum_{i=1}^{k} X_i\right] = \sum_{i=1}^{k}\mathsf{E}[X_i] = \sum_{i=1}^{k}\frac{n}{n-i+1} = n\left(\sum_{i=1}^{n}\frac{1}{i} - \sum_{i=1}^{n-k}\frac{1}{i}\right) = n\ln\frac{n}{n-k} + O(1)$ since the harmonic number $H(n) = \sum_{i=1}^{n}\frac{1}{i}$ satisfies $H(n) = \ln n + \gamma + O(1/n)$ for γ a constant. Given this, the puzzle generation cost can be calculated by counting up the other operations, and the puzzle solving cost follows because the prover must generate $\frac{1}{2}L$ index-sets in expectation and invoke hash once per index-set to solve the puzzle. $\square$

In the proof of Theorem 5.1, 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 $\mathsf{Spread}(\mathcal{I}, s)$, where $\mathcal{I}$ is a set of index-sets, to be the event that no $i \in \{1\dots n\}$ appears in s or more members of $\mathcal{I}$. Another such property is how many bits the adversary obtains in each index-set as a result of its q_{pre} queries to $\mathsf{content}$ before the puzzle is issued. We define an event $\mathsf{Remain}(\mathcal{I}, \hat{k})$ to be the event that each $I \in \mathcal{I}$ contains at least $\hat{k}$ indices that were not queried of $\mathsf{content}$ before the puzzle was issued. We abbreviate the event $\mathsf{Spread}(\mathcal{I}, s) \wedge \mathsf{Remain}(\mathcal{I}, \hat{k})$ as $\mathsf{SR}(\mathcal{I}, s, \hat{k})$.

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

Figure 14: Experiment for Theorem A.1

We begin by proving a result about the experiment in Figure 14 for a single adversary $\mathcal{A} = (\mathcal{A}_{\mathsf{pre}}, \mathcal{A}_{\mathsf{post}})$. Aside from the random oracles $\mathsf{content}$ and hash, this experiment exactly tracks the $\mathsf{verifier}$'s actions in the protocol in Figure 1. $\mathcal{A}_{\mathsf{pre}}$ represents the adversary that runs before the puzzle is issued, making q_{pre} queries to $\mathsf{content}$ and outputting state c, and $\mathcal{A}_{\mathsf{post}}$ represents the adversary that runs after the puzzle is issued, being provided as input the puzzle and c, and making q_{post} queries to $\mathsf{content}$. q_{hash} denotes the number of queries to hash made by $\mathcal{A}_{\mathsf{pre}}$ and $\mathcal{A}_{\mathsf{post}}$, combined. In the experiment, let $\mathbf{Func}(\mathsf{Dom} \rightarrow \mathsf{Rng})$ be the set of all functions with domain Dom and range Rng.

Theorem A.1 *For any s such that $1 \le s \le PL$ and any $\hat{k} \ge \log_2(q_{\mathsf{hash}} + L) + 2$,*

$$\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \le \frac{1}{L}\left(\frac{s q_{\mathsf{post}}}{\hat{k} - \log_2(q_{\mathsf{hash}} + L) - 1} + 1\right)$$

where $\mathcal{I} = \{I_1 \ldots I_L\}$ *and* $I_\ell = \{f^2_{K_2}(1) \ldots f^2_{K_2}(k)\}$ *for* $K_2 = f^1_{K_1}(\ell)$.

Proof. To prove this, we rewrite $\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right]$ as $\sum_{\ell=1}^{L} \mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \mathsf{P}\left[\hat{\ell} = \ell \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right]$, which equals

$$\frac{1}{L} \sum_{\ell=1}^{L} \mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \tag{2}$$

We now focus on bounding $\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{SR}(\mathcal{I}, s, \hat{k})\right]$ 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,

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

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

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

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

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

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

(5) and (7) follow from $\mathcal{A}$ querying $\mathsf{hash}(K_1, \ell, str)$ for only y_ℓ values str of the $2^{\hat{k}-w_\ell}$ such possible values for the $\hat{k} - w_\ell$ bits it did not retrieve from $\mathsf{content}$. 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 $2^{\hat{k}-w_\ell} - y_\ell$ values and submits this str to ans, leading to (6). Plugging these into (3), we get $\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \hat{\ell} = \ell \wedge \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \leq \min\left\{\frac{y_\ell+1}{2^{\hat{k}-w_\ell}}, 1\right\}$ and then plugging this into (2), we get

$$\begin{aligned}
\mathsf{P}\left[\mathbf{Expt}(\mathcal{A}) = 1 \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] &\leq \frac{1}{L} \sum_{\ell=1}^{L} \min\left\{\frac{y_\ell + 1}{2^{\hat{k}-w_\ell}}, 1\right\} \\
&= \frac{1}{L2^{\hat{k}}} \sum_{\ell=1}^{L} \min\left\{(y_\ell + 1)2^{w_\ell}, 2^{\hat{k}}\right\}
\end{aligned} \tag{8}$$

Consequently, we now focus on bounding

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

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

$$\sum_{\ell=1}^{m} 2^{\hat{k}-w_\ell} \le q_{\mathsf{hash}} + L < \sum_{\ell=1}^{m+1} 2^{\hat{k}-w_\ell} \tag{10}$$

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

$$\sum_{\ell=1}^{L} \min\left\{y'_\ell 2^{w_\ell}, 2^{\hat{k}}\right\} < (m+1)2^{\hat{k}} \tag{11}$$

So, to bound (9) for a given q_{post} and q_{hash}, it suffices to find $w_1 \dots w_L$ that to maximize m subject to $\sum_{\ell=1}^{L} w_\ell \le sq_{\mathsf{post}}$ and (10). For any fixed m, $\sum_{\ell=1}^{m} 2^{-w_\ell}$ is minimized by setting $w_\ell = \lceil sq_{\mathsf{post}}/m \rceil$ for $1 \le \ell \le (sq_{\mathsf{post}} \bmod m)$ and $w_\ell = \lfloor sq_{\mathsf{post}}/m \rfloor$ for $(sq_{\mathsf{post}} \bmod m) + 1 \le \ell \le m$. As such, the maximum value of m is

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

If the maximum such m divides sq_{post}, then $m2^{\hat{k}-(sq_{\mathsf{post}}/m)} \le q_{\mathsf{hash}} + L$ implies $m \le sq_{\mathsf{post}}/(\hat{k} - \log_2(q_{\mathsf{hash}} + L))$, and otherwise $m \le sq_{\mathsf{post}}/(\hat{k} - \log_2(q_{\mathsf{hash}} + L) - 1)$. Combining this with (11), we get that $\sum_{\ell=1}^{L} \min\left\{y'_\ell 2^{w_\ell}, 2^{\hat{k}}\right\} < (m+1)2^{\hat{k}} \le \left(\frac{sq_{\mathsf{post}}}{\hat{k} - \log_2(q_{\mathsf{hash}}+L) - 1} + 1\right) 2^{\hat{k}}$ and so combining this with (8) gives the result. □

For a binomially distributed random variable X with parameters m and p (denoted $X \sim \mathsf{B}(m,p)$), we have $\mathsf{P}[X = x] = \binom{m}{x} p^x (1-p)^{m-x}$ for $0 \le x \le m$, and $\mathsf{E}[X] = mp$. Let $\Psi(x,m,p) = \mathsf{P}[X \ge x]$.

Consider a set $\mathcal{P}$ of P puzzles ($|\mathcal{P}| = P$), and let puzzle p, $1 \le p \le P$, be denoted by $\langle K_1^p, \hat{h}^p \rangle$. Define sets $I_\ell^p = \{f^2_{K_2^p}(1) \dots f^2_{K_2^p}(k)\}$ for $K_2^p = f^1_{K_1^p}(\ell)$; sets $\mathcal{I}^p = \{I_1^p \dots I_L^p\}$; and set $\mathcal{I} = \bigcup_{p=1}^{P} \mathcal{I}^p$.

Lemma A.2 *For any s where $1 \le s \le PL$, $\mathsf{P}[\neg\mathsf{Spread}(\mathcal{I}, s)] \le n\Psi\left(s, PL, \frac{k}{n}\right)$.*

Proof. The number of occurrences c_i of i in $\mathcal{I}$ (i.e., $c_i = |I \in \mathcal{I} : i \in I|$) is binomially distributed, i.e., $c_i \sim \mathsf{B}(PL, k/n)$. So, $\mathsf{P}[\neg\mathsf{Spread}(\mathcal{I}, s)] = \mathsf{P}[\bigvee_{i=1}^{n} (c_i \ge s)] \le \sum_{i=1}^{n} \mathsf{P}[c_i \ge s] \le n\Psi\left(s, PL, \frac{k}{n}\right)$. □

Lemma A.3 ([39]) *Let Y be hypergeometrically distributed with parameters m (draws without replacement), N (total elements), and M (the success elements), denoted $Y \sim \mathsf{H}(m, M, N)$. If $X \sim \mathsf{B}(m, M/N)$, then $\mathsf{E}[Y] = \mathsf{E}[X] = \frac{mM}{N}$ and for any $x \geq 1 + \frac{mM}{N}$, $\mathsf{P}[Y \geq x] \leq \mathsf{P}[X \geq x]$.*

Lemma A.4 *If an adversary makes q_{pre} queries to* content *prior to puzzles $\mathcal{P}$ being issued, $\mathsf{P}\left[\neg\mathsf{Remain}(\mathcal{I}, \hat{k})\right] \leq PL\Psi\left(k - \hat{k}, k, \frac{q_{\mathsf{pre}}}{n}\right)$ for any $\hat{k} \leq k\left(1 - \frac{q_{\mathsf{pre}}}{n}\right) - 1$.*

Proof. Recall that each index set $I_\ell^p \in \mathcal{I}$ is defined as a set $\{f_K^2(1) \dots f_K^2(k)\}$ for some K. Let c_ℓ^p be the number of values $k' \in \{1 \dots k\}$ for which the adversary queries $f_K^2(k')$ prior to puzzles being issued. $\neg\mathsf{Remain}(\mathcal{I}, \hat{k})$ occurs only if $c_\ell^p \geq k - \hat{k}$ for some $I_\ell^p \in \mathcal{I}$. Because $c_\ell^p \sim \mathsf{H}(k, q_{\mathsf{pre}}, n)$, $\mathsf{P}\left[c_\ell^p \geq k - \hat{k}\right] \leq \Psi\left(k - \hat{k}, k, \frac{q_{\mathsf{pre}}}{n}\right)$ by Lemma A.3. Since there are PL index-sets in $\mathcal{I}$, the result follows. □

Proof of Theorem 5.1. Index the adversaries by $1 \dots A$ and the puzzles by $1 \dots P$. 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 s, $1 \leq s \leq PL$, and any $\hat{k} \leq k\left(1 - \frac{q_{\mathsf{pre}}}{n}\right) - 1$,

$$\begin{aligned} \mathsf{E}[S] &= \mathsf{E}\left[S \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \mathsf{P}\left[\mathsf{SR}(\mathcal{I}, s, \hat{k})\right] + \mathsf{E}\left[S \mid \neg\mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \mathsf{P}\left[\neg\mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \\ &\leq \mathsf{E}\left[S \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] + P\left(\mathsf{P}[\neg\mathsf{Spread}(\mathcal{I}, s)] + \mathsf{P}\left[\neg\mathsf{Remain}(\mathcal{I}, \hat{k})\right]\right) \end{aligned} \tag{12}$$

The result then follows by plugging in Lemmas A.2 and A.4, and the fact that for $\hat{k} \geq \log_2(q_{\mathsf{hash}} + L) + 2$,

$$\mathsf{E}\left[S \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] = \sum_{a=1}^{A} \sum_{p=1}^{P} \mathsf{P}\left[S^{ap} = 1 \mid \mathsf{SR}(\mathcal{I}, s, \hat{k})\right] \leq \frac{AP}{L}\left(\frac{sq_{\mathsf{post}}}{\hat{k} - \log_2(q_{\mathsf{hash}} + L) - 1} + 1\right)$$

by Theorem A.1. □