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

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

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

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 puzzle mechanism to make contribution-aware peer-assisted content distribution robust to such collusion. Our construction ties solving the puzzle to possession of specific content and, by issuing puzzle challenges simultaneously to all parties claiming to have that content, our mechanism prevents one content-holder from solving many others' puzzles. We prove (in the random oracle model) the security of our scheme, describe our integration of bandwidth puzzles into a media streaming system, and demonstrate the resulting attack resilience via simulations.

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., [1, 2, 3]) or priority service through server-assisted downloads (e.g., [4]), or with other mechanisms (e.g., discount coupons [4]). 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" [5,6], 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 [1, 2]. 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., [3,7,8,9]). 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., [1, 10, 2, 11]), or in P2P systems that have a distinguished node for tracking content-transfer transactions (e.g., [3,4,12]), 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 some specific 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. 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. 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 lightweight and easy to implement in peer-assisted content distribution systems. Its security analysis, however, is more subtle than the 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 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 bits, 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 find 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 also show 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.

To summarize, the contributions of this paper are: (i) the design of bandwidth puzzles (§4), a practical defense against a documented form of attack on P2P systems; (ii) analyses of our construction (in the random oracle model) that bounds the success attainable by adversaries against it (§5); (iii) implementation and evaluation of our construction in a functional streaming application (§6); and (iv) a demonstration of the benefits of puzzles on a simulated large-scale P2P streaming deployment (§7).

2 Related Work

Incentives in P2P systems: Several studies have demonstrated the limitations of P2P protocols in the presence of selfish or malicious users [13, 7]. Rewarding peer contributions can overcome these limitations (e.g., [1, 13]), but such 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, [13, 14, 15, 16]) point out the limitations of bilateral mechanisms, and make the case for *global* 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., [4]). 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., [3]). Some global contribution-awareness schemes use pricing mechanisms (e.g., [17]), some of which are theoretically collusion-resistant (e.g., [16]). However, currency management presents practical challenges for these schemes. These challenges include bootstrapping new users in a Sybil-proof manner and ensuring rapid price convergence and sufficient liquidity in the presence of system churn. Bandwidth puzzles are a lightweight alternative to provide 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 [4] or proof-of-service [18] mechanisms.

Client puzzles: Client puzzles (e.g., [19, 20, 21]) force clients to demonstrate proofs-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. Douceur [22] suggests that Sybils can be detected using simultaneous puzzle challenges similar to our work. These puzzles validate that each claimed "identity" owns some amount of computation resources. Bandwidth puzzles instead validate that each client has expended some amount of communication resources.

Proofs of data possession (PDP) and retrievability (POR): Our puzzle mechanism ties the puzzle solution to some specific content. In this respect, our construction is related to proofs of data possession (PDP) [23, 24, 25] and proofs of retrievability (POR) [26, 27, 28], that 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 PDP/POR schemes and our puzzle scheme. First, PDP/POR focus only on the interaction between a single prover and verifier, and do not deal with multiple colluding adversaries. 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. Third, PDPs/PORs presume that the verifier no longer possesses the file about which it is querying. However, many settings in which we are interested (e.g., multimedia streaming of live events) 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 untrusted *peers*, also called *provers*. Any node can be a verifier, if it can obtain the list of peers that purport to possess certain content and it has access to that content. We assume that peers report to the verifier the content they claim to have downloaded from others. P2P-assisted CDNs (e.g., [12], `www.pandonetworks.com/cdn-peering`), P2P assisted file-hosting (e.g., `www.vipeers.com`), and P2P streaming (e.g., [10, 1, 2, 29]) have a central node that can (or already does) serve this role.

Our goal 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 duration required by each prover to report its solution. We presume that the network latencies for sending puzzles and solutions between the verifier and the provers are stable over the timescales involved in puzzle solving [30]. On the basis of solution correctness and the puzzle-solving time, which it compares to a threshold θ, the verifier generates a list of peers suspected of not having the claimed content. The verifier can then take action to ensure that the uploaders in these suspicious transfers do not receive credits for them.

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 three caveats. First, 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. Second, the content must 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.[1] Third, due to churn, peers that previously downloaded content might no longer be available for solving puzzles. These peers could, however, aid collaborators that remain in the system by solving puzzles for them. Our scheme is most effective if most content exchanges for which a peer should be rewarded occur shortly after the initial distribution of the content, as would be appropriate for, e.g., streaming video of a live event. In this way, any content held by such "hidden" collaborators quickly becomes useless for solving puzzles.

4 The Construction

Let "$\leftarrow$" denote assignment; "$x \overset{R}{\leftarrow} X$" denote selection of an element from set X uniformly at random and its assignment to x; and "$||$" denote concatenation.

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.

[1] Note that this incompressibility requirement is already true for many of the popular formats (e.g., MPEG, DivX) in use today for transferring multimedia content.

Hash functions: We use two hash functions: $\mathsf{hash} : \{0,1\}^\kappa \times \{1 \ldots L\} \times \{0,1\}^k \to \{0,1\}^\kappa$ and $\mathsf{ans} : \{0,1\}^k \to \{0,1\}^\kappa$. (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 §5, we model hash as a random oracle, though collision-resistance of ans suffices.

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 $K \overset{R}{\leftarrow} \{0,1\}^\kappa$, and an oracle for a perfectly random function with the same domain and range; see [31] for a formal definition. We use families $\{f_K^1 : \{1 \ldots L\} \to \{0,1\}^\kappa\}$ and $\{f_K^2 : \{1 \ldots k\} \to \{1 \ldots n\}\}$. We require that each f_K^2 be injective, and thus that $k \le n$, where n is the content size in bits. We will discuss efficient implementations for f^2 below.

Pseudorandom functions and hash functions 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.[2] For the rest of this paper, we assume that these properties hold, ignoring events that occur with probability negligible in κ.

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

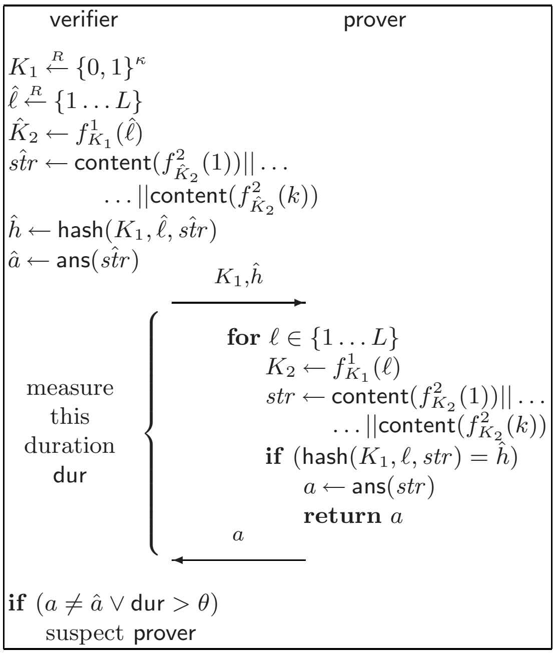

```
verifier                                             prover
$K_1 \overset{R}{\leftarrow} \{0,1\}^\kappa$
$\hat{\ell} \overset{R}{\leftarrow} \{1 \ldots L\}$
$\hat{K}_2 \leftarrow f^1_{K_1}(\hat{\ell})$
$\hat{str} \leftarrow \mathsf{content}(f^2_{\hat{K}_2}(1)) || \ldots$
        $\ldots || \mathsf{content}(f^2_{\hat{K}_2}(k))$
$\hat{h} \leftarrow \mathsf{hash}(K_1, \hat{\ell}, \hat{str})$
$\hat{a} \leftarrow \mathsf{ans}(\hat{str})$
                     $K_1, \hat{h}$  →
 measure this        for $\ell \in \{1 \ldots L\}$
 duration dur            $K_2 \leftarrow f^1_{K_1}(\ell)$
                         $str \leftarrow \mathsf{content}(f^2_{K_2}(1)) || \ldots$
                                 $\ldots || \mathsf{content}(f^2_{K_2}(k))$
                         if $(\mathsf{hash}(K_1, \ell, str) = \hat{h})$
                             $a \leftarrow \mathsf{ans}(str)$
                             return $a$
                     ←  $a$
if $(a \ne \hat{a} \vee \mathsf{dur} > \theta)$
    suspect prover
```

Fig. 1. One bandwidth puzzle

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

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 Fig. 1. First, it randomly selects a key K_1 for the family f^1 and an index $\hat{\ell} \overset{R}{\leftarrow} \{1 \ldots L\}$ 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 $\mathsf{ans}(\mathit{str})$ for the string str satisfying $\hat{h} = \mathsf{hash}(K_1, \hat{\ell}, \mathit{str})$, 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).

Finally, a subtle but important implementation challenge arises for f^2, because our security analysis in §5 requires that f^2 be injective. 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., [32]). 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 $\{f^3_K : \{1, 2, \ldots\} \rightarrow \{1 \ldots n\}\}$, 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:

Theorem 1. *The construction of Fig. 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.*

Proof. The result follows by a "coupon collector" analysis. When generating $f^2_{K_2}(1) \ldots f^2_{K_2}(k)$ 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.

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 §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 [31], and that ans is a collision-resistant hash function. 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. We model a prover's retrieval of content bits as calls to a random oracle $\mathsf{content} : \{1 \dots n\} \to \{0, 1\}$. As discussed in §3, our construction requires 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.

Theorem 2. *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* $1 \le s \le PL$ *and* $\log_2(q_{\mathsf{hash}} + L) + 2 \le \hat{k} \le k\left(1 - \frac{q_{\mathsf{pre}}}{n}\right) - 1$*, the expected number of puzzles that these adversaries can solve collectively is at most*

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

where $\Psi(x, m, p) = \mathsf{P}\left[X \ge x\right]$ *for a binomially distributed r.v.* $X \sim \mathsf{B}(m, p)$*.*

The proof of this result is too lengthy to include here; the interested reader is referred to our companion technical report [33]. Very briefly, the second term of this sum accounts for the possibility that some $i \in \{1 \dots n\}$ appears in s or more index-sets, and the third term accounts for the possibility that the adversaries queried $k - \hat{k}$ or more indices in some index-set before the puzzles were issued.

The first term, then, bounds the number of puzzles the adversaries solve in expectation when neither of these events occur.

To see a consequence of Theorem 2, 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 that $P^2 L\Psi\left(k - \hat{k}, k, \frac{Aq_{\mathsf{pre}}}{n}\right) \to 0$ as $n \to \infty$. Setting $s = (1 + \delta')\frac{PLk}{n}$ for $\delta' > 0$ implies $Pn\Psi\left(s, PL, \frac{k}{n}\right) \to 0$ as $n \to \infty$. For this value of s, Theorem 2 implies that $q_{\mathsf{post}} = \Omega(n)$ 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.

Examples of Theorem 2 for different values of A and n are shown in Fig. 2, which plots the minimum of P and that bound 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 Fig. 2.

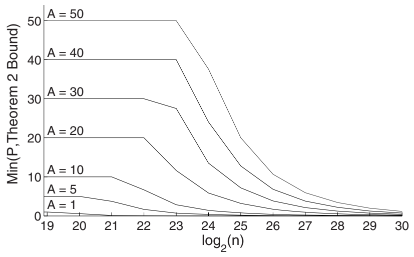


Fig. 2. An example of Theorem 2

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 [10, 1, 2, 29]. 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 [35] 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[3] to 50 seed clients[4] over a 100Mb/s network. We also 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 Fig. 2 are representative for our experiments.

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

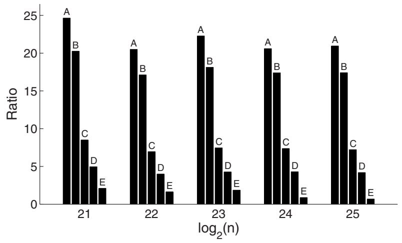


Fig. 3. Ratio of 95th percentile puzzle-solving time for Class-X machine (X $\in$ {A, B, C, D, E}) to 50th percentile puzzle-solving time for Class-E machine during live streaming experiments

Fig. 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. Fig. 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. Designing memory-bound extensions of our scheme to reduce the variability in solving time across different classes of machines [36, 21] is an interesting avenue for future work.

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

[4] As a point of comparison, the server in the popular P2P streaming system PPLive supports 25 seed clients at 400Kbps [11].

Having chosen to focus on machine classes C, D, and E, we further narrow our attention to puzzles for each $n = 2^{23}$ bits for the rest of our evaluation.

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. Fig. 4 shows the distribution of jitter of the media stream at clients for a duration including 100 puzzle challenges, for different machine classes. Fig. 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.

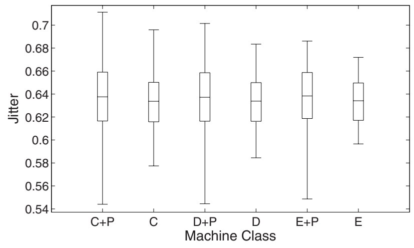
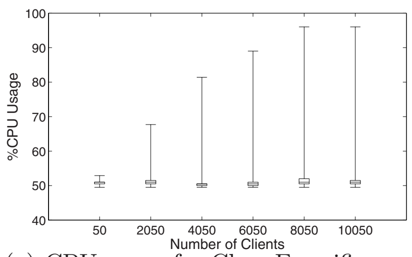


Fig. 4. Jitter per machine class. "+P" indicates with puzzles.

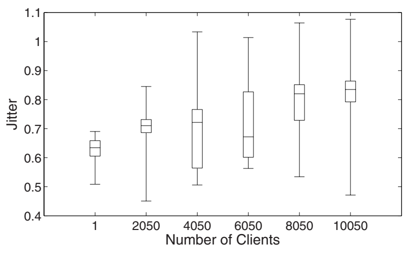
Verifier Scalability: To test scalability, we fixed the number of clients to which a Class E server streams content at 50, but had it simultaneously generate and send puzzles to a number of clients (in addition to these 50) ranging from 0 to 10000. Due to limits on the number of available Emulab computers,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



(a) CPU usage for Class-E verifier node



(b) Jitter for one Class-E client

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

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 Fig. 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 more slack.

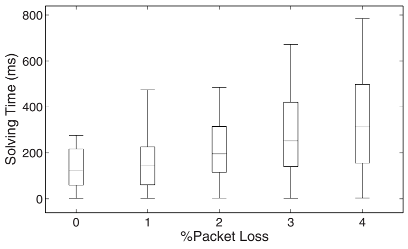


Fig. 6. Puzzle-solving time for a Class-E client as a function of packet loss

7 Benefits in a Peer-Assisted Streaming System

In this section, we show via simulation the benefits of using bandwidth puzzles in a contribution-aware peer-assisted streaming application (e.g., [10, 1, 2]).

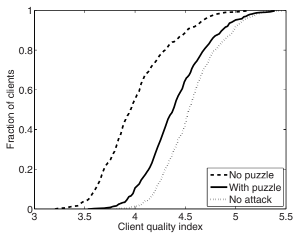
Streaming Model: We assume that the multimedia stream is divided into discrete epochs of size 1000 units of simulation time where each unit corresponds to 100ms of real time. The content within each epoch is divided into suitably encoded *stripes* [10,1]. This encoding (e.g., [37]) has the property that 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. Each stripe is broken into 1MB *chunks* and peers download the chunks corresponding to each stripe using a suitable lookup mechanism.

Incentive Mechanism: We use an incentive scheme similar to Maze [3]. The streaming server, to which peers authenticate and periodically report transactions on a per-chunk basis, maintains a per-peer "points system". Each peer earns 1.5 points for every chunk uploaded and consumes 1 point per chunk downloaded. 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.) Free-riders, i.e., with zero points, are denied service.

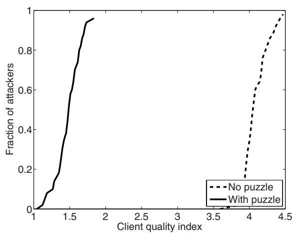
Adding bandwidth puzzles: In traditional contribution-aware systems, the server debits points from the downloader and credits points to the uploader on receiving a transaction report. In a system with bandwidth puzzles, handling transactions is slightly different. The server debits points from the downloader's account as before. However, it does not immediately credit the uploader for the transaction. Rather, at the end of each epoch, the server issues simultaneous puzzle challenges on a per-chunk basis in the role of the verifier; i.e., it iterates through the chunks for this epoch one by one, and challenges the clients that claimed to have received this chunk in the epoch. Upload credits are granted when the corresponding downloaders correctly answer their puzzle challenges.[5]

Attack Model: We specify attacks as a *collusion graph*, where each vertex is a malicious peer (actual or Sybil node). Each directed edge $x \rightarrow y$ represents an *fake uploader* relationship: peer x reports "fake" transactions on behalf of peer y, i.e., x requests the server to credit y for uploads even though y spends no bandwidth for the transfer. Each such x periodically reports fake transactions to the server in addition to its legitimate transactions (if any). 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 seeds (higher priority service). For example, in a Star(200,19) graph, there are 200 nodes in the graph, organized in 10 "star" graphs. Each star has 19 leaf nodes representing the fake (Sybil) identities and the actual attacker is the center of the star. To model attackers' responses to puzzle challenges, we assume that a puzzle sent to a peer who does not have the

[5] 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 §2.



(a) Legitimate clients



(b) Attackers

Fig. 7. Benefits in a P2P streaming system

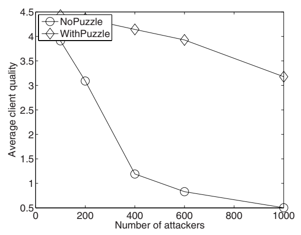


Fig. 8. Varying the number of attackers

content or to a fake peer is solved with probability 0.1. In the Star(200,19) case, this means that in expectation $19 \times 0.1 = 1.9$ fake transactions get validated.[6]

Simulation Framework: We implemented an event-driven simulator modeling chunk exchanges, transaction reports, and puzzle challenges. We do not model network congestion effects and assume that the only bandwidth bottleneck is the upstream bandwidth bottleneck of the peers. The download requests at each peer are queued based on the requester's points as described above and served one at a time without preemption. 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. All exchanges and transaction reports occur at a 1MB chunk granularity.

Performance Benefits: We define the *user quality* to be the average number of stripes received by a client per epoch in the streaming session. Fig. 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 Fig. 7(b), there is more than $2\times$ reduction in the median attacker quality when bandwidth puzzles are used. Fig. 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.

[6] Since 190 identities are fake, the attackers' resources correspond to $A = 10$. If the verifier issues puzzles per chunk ($\log_2(n) \approx 23$), the value of the bound in Theorem 2 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 Fig. 2) is consistent with setting the puzzle solving probability to be 0.1.

8 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 quantify 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 simulation 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 the anonymous reviewers for their comments. This work was supported in part by NSF awards CT-0756998, CNS-0326472, and ANI-0331653, and Florida State University CRC award PG-022684.

References

1. Sung, Y., Bishop, M., Rao, S.: Enabling Contribution Awareness in an Overlay Broadcasting System. In: Proc. ACM SIGCOMM (2006)
2. Purandare, D., Guha, R.: BEAM: An Efficient Framework for Media Streaming. In: Proc. IEEE LCN (2006)
3. Lian, Q., Zhang, Z., Yang, M., Zhao, B.Y., Dai, Y., Li, X.: An empirical study of collusion behavior in the Maze P2P file-sharing system. In: Proc. ICDCS (2007)
4. Sirivianos, M., Park, J.H., Yang, X., Jarecki, S.: Dandelion: Cooperative Content Distribution with Robust Incentives. In: Proc. USENIX ATC (2007)
5. Dellarocas, C.: Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In: Proc. ACM EC (2000)
6. Bhattacharjee, R., Goel, A.: Avoiding ballot stuffing in eBay-like reputation systems. In: Proc. ACM SIGCOMM P2P-ECON (2005)
7. Sirivianos, M., Park, J.H., Chen, R., Yang, X.: Free-riding in BitTorrent networks with the large view exploit. In: Proc. IPTPS (2007)
8. Liogkas, N., Nelson, R., Kohler, E., Zhang, L.: Exploiting BitTorrent for fun (but not profit). In: Proc. IPTPS (2006)
9. Adar, E., Huberman, B.A.: Free riding on Gnutella. First Monday 5 (2000)
10. Castro, M., et al.: SplitStream: High-bandwidth multicast in a cooperative environment. In: Proc. ACM SOSP (2003)
11. Huang, G.: Keynote: Experiences with PPLive. In: Proc. ACM SIGCOMM P2P-TV Workshop (2007)
12. Freedman, M.J., Freudenthal, E., Mazieres, D.: Democratizing content publication with Coral. In: Proc. NSDI (2004)
13. Feldman, M., Lai, K., Stoica, I., Chuang, J.: Robust Incentive Techniques for Peer-to-Peer Networks. In: Proc. ACM EC (2004)

14. Piatek, M., Isdal, T., Krishnamurthy, A., Anderson, T.: One hop reputations for peer to peer file sharing workloads. In: Proc. NSDI (2008)
15. Lai, K., Feldman, M., Stoica, I., Chuang, J.: Incentives for cooperation in peer-to-peer networks. In: Proc. P2P Econ (2004)
16. Aperjis, C., Freedman, M.J., Johari, R.: Peer-Assisted Content Distribution with Prices. In: Proc. CoNeXT (2008)
17. Belenkiy, M., et al.: Making P2P accountable without losing privacy. In: Proc. ACM WPES (2007)
18. Li, J., Kang, X.: Proof of service in a hybrid P2P environment. In: Proc. ISPA Workshops (2005)
19. Dwork, C., Naor, M.: Pricing via processing, or, combatting junk mail. In: Proc. CRYPTO (1993)
20. Juels, A., Brainard, J.: Client puzzles: A cryptographic defense against connection depletion attacks. In: Proc. NDSS (1999)
21. Dwork, C., Goldberg, A., Naor, M.: On memory-bound functions for fighting spam. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 426–444. Springer, Heidelberg (2003)
22. Douceur, J.: The Sybil attack. In: Proc. IPTPS (2002)
23. Ateniese, G., et al.: Provable data possession at untrusted stores. In: Proc. ACM CCS (2007)
24. Filho, D.L.G., Barreto, P.S.L.M.: Demonstrating data possession and uncheatable data transfer (2006), http://eprint.iacr.org/2006/150.pdf
25. Ateniese, G., Pietro, R.D., Mancini, L.V., Tsudik, G.: Scalable and Efficient Provable Data Possession (2008), http://eprint.iacr.org/2008/114.pdf
26. Juels, A., Kaliski Jr., B.S.: PORs: Proofs of retrievability for large files. In: Proc. ACM CCS (2007)
27. Bowers, K., Juels, A., Oprea, A.: Proofs of Retrievability: Theory and Implementation (2008), http://eprint.iacr.org/2008/175.pdf
28. Shacham, H., Waters, B.: Compact Proofs of Retrievability (2008), http://eprint.iacr.org/2008/073.pdf
29. Yin, H., et al.: Design and Deployment of a Hybrid CDN-P2P System for Live Video Streaming: Experiences with LiveSky. In: Proc. ACM Multimedia (2009)
30. Zhang, Y., Duffield, N., Paxson, V., Shenker, S.: On the Constancy of Internet Path Properties. In: Proc. IMW (2001)
31. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM 33(4), 792–807 (1984)
32. Black, J., Rogaway, P.: Ciphers with arbitrary finite domains. In: Preneel, B. (ed.) CT-RSA 2002. LNCS, vol. 2271, pp. 114–130. Springer, Heidelberg (2002)
33. Reiter, M.K., Sekar, V., Spensky, C., Zhang, Z.: Making contribution-aware peer-assisted content distribution robust to collusion using bandwidth puzzles. Technical Report CMU-CS-09-136, Carnegie Mellon University (2009)
34. Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A transport protocol for real-time applications. IETF RFC 3550 (July 2003)
35. White, B., et al.: An Integrated Experimental Environment for Distributed Systems and Networks. In: Proc. OSDI (2002)
36. Abadi, M., Burrows, M., Manasse, M., Wobber, T.: Moderately hard, memory-bound functions. ACM TOIT 5, 299–327 (2005)
37. Goyal, V.K.: Multiple description coding: Compression meets the network. IEEE Signal Processing Magazine, 74–93 (September 2001)