A New Bound on the Performance of the Bandwidth Puzzle

Zhenghao Zhang
Computer Science Department
Florida State University
Tallahassee, FL 30306, USA
zzhang@cs.fsu.edu

***Abstract*—A bandwidth puzzle was recently proposed to defend against colluding adversaries in peer-to-peer networks. The colluding adversaries do not do actual work but claim to have uploaded contents for each other to gain free credits from the system. The bandwidth puzzle guarantees that if the adversaries can solve the puzzles, they must have spent substantial bandwidth, the size of which is comparable to the size of the contents they claim to have uploaded for each other. Therefore, the puzzle discourages the collusion. In this paper, we study the performance of the bandwidth puzzle and give a lower bound on the *average* number of bits the adversaries must receive to be able to solve the puzzles with a certain probability. We show that our bound is tight in the sense that there exists a strategy to approach this lower bound asymptotically within a small factor. The new bound gives better security guarantees than the existing bound, and can be used to guide better choices of puzzle parameters to improve the system performance.**

I. Introduction

A key problem in peer-to-peer (p2p) based content sharing is the incentive for peers to contribute bandwidth to serve other peers [14]. Without a robust incentive mechanism, peers may choose not to upload contents for other peers, causing the entire system to fail. In many applications, a peer's contribution is measured by the number of bits it uploaded for other peers. It is difficult to measure the contribution because peers may collude with each other to get free credits. For example, if Alice and Bob are friends, Alice, without actually uploading, may claim that she has uploaded a certain amount of bits to Bob. Bob, when asked about this claim, will attest that it is true because he is Alice's friend. Therefore, Alice gets free credits.

With the current Internet infrastructure, such collusions are difficult to detect, because the routers do not keep records of the traffic. Recently, a bandwidth puzzle scheme has been proposed solve this problem [12]. In the bandwidth puzzle scheme, a central credit manager, called the *verifier*, is assumed to exist in the network. The verifier issues puzzles to suspected nodes, called *provers*, to verify whether the claimed transactions are true. To be more specific, when the verifier suspects a set of provers for certain transactions, it issues puzzles *simultaneously* to all the involved provers, and asks them to send back answers within a time threshold. The puzzle's main features are: 1) it takes time to solve a puzzle and 2) a puzzle can be solved only if the prover has access to the contents. To illustrate the basic idea of the puzzle, consider the previous simple example with Alice and Bob. The verifier issues two puzzles, one to Alice and one to Bob. As Alice did not upload the content to Bob, Alice has the content but not Bob. When received the puzzles, Alice can solve hers and send the answer to the verifier before the threshold but not Bob. Neither can Bob ask help from Alice, because Alice cannot solve two puzzles within the threshold. Given this, Bob will fail to reply with the answer of the puzzle and the verifier will know that the transaction did not take place.

The bandwidth puzzle is most suited for live video broadcast applications, where fresh contents are generated constantly [12]. The verifier can naturally reside in the source node of the video, and the puzzle is based on the unique content currently being broadcast, such that there are no existing contents downloaded earlier that can be used to solve the puzzles. The construction of bandwidth puzzle is simple and based only on hash functions and pseudorandom functions. In [12], the puzzle scheme was implemented and incorporated into a p2p video distributing system, and was shown to be able to limit collusions significantly. An upper bound was also given for the expected number of puzzles that can be solved given the limit of the number of bits received among the adversaries. However, the bound is "loose in several respects," as stated by the authors, because its dominating term is quadratic to the number of adversaries such that it deteriorates quickly as the number of adversaries increases. In this paper, we give a much improved bound on the performance of the puzzle. The new bound gives the *average* number of bits the adversaries must have received if they can solve the puzzles with a certain probability. As we will prove, the average number of bits the adversaries receive is linear to the number of adversaries. It is also asymptotically tight in the sense that there exists a strategy

n	The number of bits in the content
k	The number of indices in an index set
L	The number of index sets in a puzzle
z	The number of puzzles sent to a prover
θ	The time threshold to solve the puzzles

TABLE I
LIST OF PUZZLE PARAMETERS

that achieves this bound asymptotically within a small factor. The improved bound leads to more relaxed constraints on the choice of puzzle parameters, which should in turn improve the system performance.

The rest of this paper is organized as follows. Section II describes the construction of the puzzle. Section III gives the proof of the new bound. Section IV discusses related works. Section V concludes the paper.

II. THE CONSTRUCTION

In this section, we describe the construction of the puzzle. The puzzle construction is largely the same as [12] except one difference: allowing repeated indices in one index set (the definition of index set will be given shortly), which simplifies the puzzle construction. We first give a high-level overview of the puzzle construction as well as introducing some notations. The main parameters of the puzzle are listed in Table I.

A. A High-level Description

The content being challenged is referred to simply as *content*. There are n bits in the content, each given a unique index. An *index set* is defined as k ordered indices chosen from the n indices. Each index set defines a string denoted as str, called the *true string* of this index set, which is obtained by reading the bits in the content according to the indices. str can be hashed using a hash function denoted as hash, and the output is referred to as the hash of the index set. To construct a puzzle, the verifier needs L index sets denoted as $I_1, \ldots, I_L$, where an index set is obtained by randomly choosing the indices, allowing repeat. The verifier randomly chooses one index set among the L index sets, denoted as $I_{\hat{\ell}}$, called the *answer index set*. It uses hash to get the hash of $I_{\hat{\ell}}$, denoted as $\hat{h}$, which is called the *hint* of the puzzle. The puzzle is basically the L index sets and $\hat{h}$. When challenged with a puzzle, the prover should prove that it knows which index set hashes into $\hat{h}$, by presenting another hash of $I_{\hat{\ell}}$ generated by hash function ans. The purpose of using ans is to reduce the communication cost, as $str_{\hat{\ell}}$ may be long. The verifier may issue z puzzles to the prover and the prover has to solve the all puzzles before a time threshold θ.

The strengths of the puzzle are: 1) a prover has to know the content, otherwise it cannot get the true strings of the index sets, and 2) even if the prover knows the content, it still has to spend time and try different index sets until it finds an index set with the same hash as the hint, refereed to as a $\mathsf{confirm}$ event, because the hash function is one-way. In practice, the verifier need not generate all index sets; it need only generate and find the hash of the answer index set. The verifier should not send the L index sets to the prover because this requires a large communication cost; instead, the verifier and the prover can agree on the same pseudorandom functions to generate the index sets and the verifier sends only a key for the pseudorandom functions. Therefore, this construction has low computation cost and low communication cost.

As a example, suppose $n = 8$ and the content is 00110101. Suppose $k = 4$, $L = 3$, and the three index sets in the puzzle are $I_1 = \{5, 3, 7, 0\}$, $I_2 = \{1, 2, 6, 3\}$, and $I_3 = \{2, 3, 5, 3\}$. Correspondingly, $str_1 = 1110$, $str_2 = 0101$ and $str_3 = 1111$. Suppose the verifier chooses $\hat{\ell} = 1$. Suppose hash is simply the parity bit of the string, such that $\hat{h} = 1$. The prover receives the hint and generates the three index sets, and finds that only I_1 has parity bit 1. Suppose ans is simply the parity bit of every pair of adjacent bits. The prover presents '01' which proves that it knows I_1 is the answer index set.

B. Detailed Puzzle Construction

In the construction, it is assumed that the keys of the pseudorandom functions and the output of the hash functions are both κ bits. In practice, $\kappa = 160$ suffices.

Pseudorandom functions are used to generate the index sets. A pseudorandom function family $\{f_K\}$ is a family of functions parameterized by a secret key. Roughly speaking, once initialized by a key, a pseudorandom function generates outputs that are indistinguishable from true random outputs. Two pseudorandom function families are used: $\{f^1_K : \{1, \ldots, L\} \rightarrow \{0, 1\}^\kappa\}$ and $\{f^2_K : \{1, \ldots, k\} \rightarrow \{1, \ldots, n\}\}$.

Two hash functions are used in the construction, hash and ans. hash is used to get the hint. It actually hashes the concatenation of a κ-bit key, a number in the range of $[1, L]$, and a k-bit string into κ-bits: $\{0, 1\}^\kappa \times \{1, \ldots, L\} \times \{0, 1\}^k \rightarrow \{0, 1\}^\kappa$. To prove the security of the puzzle, hash is modeled as a random oracle [4]. The other hash function is $\mathsf{ans} : \{0, 1\}^k \rightarrow \{0, 1\}^\kappa$. For ans, only collision-resistance is assumed.

As mentioned earlier, a puzzle consists of the hint $\hat{h}$ and L index-sets. The verifier first randomly picks a κ-bit string as key K_1. Then it randomly picks a number $\hat{\ell}$ from $[1, L]$ as the index of the answer index set. With K_1 and $\hat{\ell}$, it generates $K^{\hat{\ell}}_2 \leftarrow f^1_{K_1}(\hat{\ell})$. $K^{\hat{\ell}}_2$ is used as the key for $f^2_{K_2}$ to generate the indices in the answer index set: $I_{\hat{\ell}} = \{f^2_{K^{\hat{\ell}}_2}(1) \ldots f^2_{K^{\hat{\ell}}_2}(k)\}$. The verifier then finds $str_{\hat{\ell}}$. It then uses the concatenation of K_1, $\hat{\ell}$, and $str_{\hat{\ell}}$ as the input to hash and uses the output as $\hat{h}$: $\hat{h} \leftarrow \mathsf{hash}(K_1, \hat{\ell}, str_{\hat{\ell}})$. Including K_1 and $\hat{\ell}$ ensures that the results of one puzzle-solving process cannot be used for another puzzle, regardless of the content, k, and L. The prover can generate index sets in the same way as the verifier generates the answer index set, and can compare the hash of the index sets with the hint until a $\mathsf{confirm}$ is found. When the prover finds a $\mathsf{confirm}$ upon string str_ℓ, it returns $\mathsf{ans}(str_\ell)$.

III. THE SECURITY BOUND

In this section, we derive the new bound for the bandwidth puzzle. Although the puzzle is designed to defend against colluding adversaries, we begin with the simple case when there is only one adversary given only one puzzle, because the proof for this simple case can be extended to the case when multiple adversaries are given multiple puzzles.

q_{hash}	The number of hash queries allowed, determined by θ
Ω	A special oracle for hash and content queries
V	The maximum number of missed bits
δ	A positive number determined by puzzle parameters

TABLE 2
LIST OF NOTATIONS IN THE PROOF

A. Single Adversary with a Single Puzzle

Consider a single adversary challenged with one puzzle. We begin with assumptions and definitions. Some key proof parameters and notations are listed in Table 2.

1) Assumptions and Definitions: In the proof, we model hash and ans as random oracles and refer to them as the *hash oracle* and the *answer oracle*, respectively. Obtaining a bit in the content is also modeled as making a query to the *content oracle* denoted as content. The adversary is given access to hash, ans, and content. To model the computational constraint of the prover in the limited time θ allowed to solve the puzzle, we assume the number of queries to hash is no more than q_{hash}. To ensure that honest provers can solve the puzzle, $q_{\mathsf{hash}} \geq L$. However, we do not assume any limitations on the number of queries to content and ans. We refer a query to content as a *content query* and a query to hash a *hash query*. We use $\mathcal{A}$ to denote the algorithm adopted by the adversary.

In our proof, we define a special oracle, Ω, as an oracle that answers two kinds of queries, both the content query and the hash query. Let $\mathcal{B}$ be an algorithm for solving the puzzle, when given access to the special oracle Ω and the answer oracle ans. If $\mathcal{B}$ makes a content query, Ω simply replies with the content bit. In addition, it keeps the history of the content queries made. When $\mathcal{B}$ makes a hash query to Ω for a string, if it has made content queries for more than $k - V$ bits in this string, we say the hash query is *informed* and *uninformed* otherwise, where V is a proof parameter much smaller than k. If $\mathcal{B}$ makes an informed hash query for I_ℓ, Ω replies with the hash of I_ℓ; otherwise, it returns $\emptyset$. In addition, if $\mathcal{B}$ makes more than L hash queries for the puzzle, Ω will not answer further hash queries.

2) Problem Formalization: The questions we seek to answer is: given q_{hash}, if the adversary has a certain advantage in solving the puzzle, how many content queries it must make to content *on average*? In the context of p2p content distribution, this is analogous to giving a lower bound on the average number of bits a peer must have downloaded if it can pass the puzzle challenge with a certain probability. Note that we emphasize on the average number of bits because a deterministic bound may be trivial: if the adversary happens to pick the answer index set in the first attempt of hash queries, only k content queries are needed. However, the adversary may be lucky once but unlikely to be always lucky. Therefore, if challenged with a large number of puzzles, the average number of queries it makes to content must be above a certain lower bound, which is the bound we seek to establish.

3) Proof Sketch: A sketch of our proof is as follows. As it is difficult to derive the optimal algorithm the adversary may adopt, our proof is "indirect." That is, by using Ω, we introduce a simplified environment which is easier to reason about. We show that given an algorithm for the real environment, an algorithm for the simplified environment can be constructed with performance close to the algorithm for the real environment. This provides a link between the simplified environment and the real environment: knowing the bound for the former, the bound for the latter is a constant away. We establish the performance bound of the optimal algorithm in the simplified environment, by showing that to solve the puzzle with certain probability, an algorithm must make a certain number of informed hash queries to Ω and the average number of unique indices in the informed queries, i.e., the number of content queries, is bounded.

4) Proof Details: Given any algorithm $\mathcal{A}$ the adversary may adopt, we construct an algorithm $\mathcal{B}_\mathcal{A}$ that employs $\mathcal{A}$ and implements oracle queries for $\mathcal{A}$. $\mathcal{B}_\mathcal{A}$ terminates when $\mathcal{A}$ terminates, and returns what $\mathcal{A}$ returns. When $\mathcal{A}$ makes a query, $\mathcal{B}_\mathcal{A}$ replies as follows:

```
Algorithm 1 $\mathcal{B}_\mathcal{A}$ answers oracle queries for $\mathcal{A}$
1: When $\mathcal{A}$ makes a query to content, $\mathcal{B}_\mathcal{A}$ makes the same content query to $\Omega$ and **returns** the result to $\mathcal{A}$.
2: When $\mathcal{A}$ makes a query to ans, $\mathcal{B}_\mathcal{A}$ makes the same query to ans and **returns** the result to $\mathcal{A}$.
3: When $\mathcal{A}$ makes a query to hash for $I_\ell$:
     1) $\mathcal{B}_\mathcal{A}$ checks whether $\mathcal{A}$ has made exactly the same query before. If yes, it **returns** the same answer as the last time.
     2) $\mathcal{B}_\mathcal{A}$ checks whether there are no less than $V$ bits in $I_\ell$ that have not been queried. If yes, it **returns** a random string.
     3) $\mathcal{B}_\mathcal{A}$ checks whether it has made a hash query for $I_\ell$ before. If no, $\mathcal{B}_\mathcal{A}$ makes a hash query to $\Omega$. If confirm is obtained upon this query, $\mathcal{B}_\mathcal{A}$ knows that $I_\ell$ is the answer index set, and sends content queries $\Omega$ to get the remaining bits in $I_\ell$.
     4) If $I_\ell$ is not the answer index set, $\mathcal{B}_\mathcal{A}$ **returns** a random string.
     5) If the string $\mathcal{A}$ submitted is the true string of $I_\ell$, $\mathcal{B}_\mathcal{A}$ **returns** the hash of $I_\ell$.
     6) $\mathcal{B}_\mathcal{A}$ **returns** a random string.
```

Let $\omega()$ denote the average number of bits received by an algorithm, where the average is taken over the random choices of the algorithm and the randomness of the puzzle. We have

Theorem 3.1: Let $C_\mathcal{A}$ be the event that $\mathcal{A}$ returns the correct answer when $\mathcal{A}$ is interacting directly with content, hash and ans. Let $C_{\mathcal{B}_\mathcal{A}}$ be the event that $\mathcal{B}_\mathcal{A}$ returns the correct answer, when $\mathcal{B}_\mathcal{A}$ is interacting with Ω and ans. Then,

$$\mathsf{P}\left[C_{\mathcal{B}_\mathcal{A}}\right] \geq \mathsf{P}\left[C_\mathcal{A}\right] - \frac{q_{\mathsf{hash}}}{2^V},$$

and

$$\omega[\mathcal{B}_\mathcal{A}] \leq \omega[\mathcal{A}] + \frac{Lkq_{\mathsf{hash}}}{2^V} + V.$$

Proof: In our construction, $\mathcal{B}_{\mathcal{A}}$ employs $\mathcal{A}$, and answers oracle queries for $\mathcal{A}$. Denote the random process of $\mathcal{A}$ when it is interacting directly with content, hash and ans as W, and denote the random process of $\mathcal{A}$ when it is interacting with the oracles implemented by $\mathcal{B}_{\mathcal{A}}$ as W'. We prove that W and W' will progress in the same way statistically with only one exception, while the probability of this exception is bounded.

First, we note that when $\mathcal{A}$ makes a query to content or ans, $\mathcal{B}_{\mathcal{A}}$ simply gives the query result, therefore the only case needs to be considered is when $\mathcal{A}$ makes a query to hash. When $\mathcal{A}$ makes a query for I_ℓ to hash,

- If there are still no less than V unknown bits in this index set, $\mathcal{B}_{\mathcal{A}}$ will simply return a random string, which follows the same distribution as the output of the hash modeled as a random oracle. If $\ell \neq \hat{\ell}$, such a query will not result in a confirm, and this will have same effect on the progress of the algorithm statistically as when $\mathcal{A}$ is making a query to hash. However, if $\ell = \hat{\ell}$, it could happen that $\mathcal{A}$ is making a query with the true string. In this case, the exception occurs. That is, W' will not terminate, but W will terminate with the correct answer to the puzzle. However, the probability of this exception is bounded from the above by $\frac{q_{\mathsf{hash}}}{2^V}$, because if no less than V bits are unknown, the probability of making a hash query with the true string is no more than $\frac{q_{\mathsf{hash}}}{2^V}$.
- If $\mathcal{B}_{\mathcal{A}}$ has made enough content queries for this index set, $\mathcal{B}_{\mathcal{A}}$ checks whether it has made hash query for this index set before. If no, $\mathcal{B}_{\mathcal{A}}$ makes the hash query, and if a confirm is obtained, $\mathcal{B}_{\mathcal{A}}$ knows that this is the answer index set and get the possible remaining bits in it; otherwise $\mathcal{B}_{\mathcal{A}}$ knows that it is not the answer index set. If I_ℓ is not the answer index set, $\mathcal{B}_{\mathcal{A}}$ will simply return a random string, which will have the same effect statistically on the progress of $\mathcal{A}$ as when $\mathcal{A}$ is interacting with hash. If I_ℓ is the answer index set, $\mathcal{B}_{\mathcal{A}}$ checks whether $\mathcal{A}$ is submitting the true string, and returns the true hash if yes and a random string otherwise. This, clearly, also has the same effect statistically of the progress of $\mathcal{A}$ as when $\mathcal{A}$ is interacting with hash.

From the above discussion, we can see that $\mathsf{P}[C_{\mathcal{B}_{\mathcal{A}}}]$ is no less than $\mathsf{P}[C_{\mathcal{A}}]$ minus the probability of the exception. Therefore, the first half of the theorem is proved. We can also see that if the exception occurs, $\mathcal{B}_{\mathcal{A}}$ makes at most Lk more content queries than $\mathcal{A}$. If the exception does not occur, $\mathcal{B}_{\mathcal{A}}$ receives at most V bits than $\mathcal{A}$ it encapsulates, and therefore at most V bits more than $\mathcal{A}$ on average when $\mathcal{A}$ is interacting directly with content, hash and ans. ■

Theorem 3.1 allows us to establish a connection between the "real" puzzle solver and the puzzle solver interacting with Ω. The advantage of introducing Ω is that a good algorithm will not send any uninformed queries to Ω, because it will get no information from such queries. If there is a bound on the number of hash queries, which are all informed, it is possible to establish a lower bound on the number of unique indices involved in such queries, with which the lower bound of the puzzle can be established. It is difficult to establish such bound based on hash directly because hash answers any queries. Although some queries are "more informed" than others, all queries have non-zero probabilities to get a confirm. The next theorem establishes the lower bound on the expected number of informed hash queries to achieve a given advantage by an optimal algorithm interacting with Ω.

Theorem 3.2: Suppose $\mathcal{B}$ is an optimal algorithm for solving the puzzle when interacting with Ω. If $\mathcal{B}$ solves the puzzle with probability no less than ϵ, on average, the number of informed hash queries it makes is no less than $\frac{(\epsilon - \frac{1}{2^V})(L+1)}{2}$.

Proof: Let correct denote the event that $\mathcal{B}$ returns the correct answer. Note that

$$\begin{aligned}\mathsf{P}[\mathsf{correct}] &= \mathsf{P}[\mathsf{correct} \mid \mathsf{confirm}]\,\mathsf{P}[\mathsf{confirm}] \\ &\quad + \mathsf{P}[\mathsf{correct} \mid \neg\mathsf{confirm}]\,\mathsf{P}[\neg\mathsf{confirm}] \\ &= \mathsf{P}[\mathsf{confirm}] \\ &\quad + \mathsf{P}[\mathsf{correct} \mid \neg\mathsf{confirm}]\,\mathsf{P}[\neg\mathsf{confirm}] \\ &\leq \mathsf{P}[\mathsf{confirm}] + \mathsf{P}[\mathsf{correct} \mid \neg\mathsf{confirm}] \\ &\leq \mathsf{P}[\mathsf{confirm}] + \frac{1}{2^V}\end{aligned}$$

Note that $\mathsf{P}[\mathsf{correct} \mid \neg\mathsf{confirm}] \leq \frac{1}{2^V}$ because if the algorithm returns the correct answer, it must have the true string of the answer index set, since ans is collision-resistant. If a confirm was not obtained, the answer index set is missing no less than V bits, since otherwise an optimal algorithm should make query which will result in a confirm. Therefore, the probability that the algorithm can obtain the true string of the answer index set is no more than $\frac{1}{2^V}$. Note that hash queries to Ω will not help in the guessing of the true string, because Ω is aware of the number of missing bits and will not reply with any information. Therefore, any algorithm that achieves advantage ϵ in solving the puzzle must have an advantage of no less than $\epsilon - \frac{1}{2^V}$ to get confirm.

Let P_1 be the probability that $\mathcal{B}$ makes no hash query and let P_i be the probability that $\mathcal{B}$ stops making hash queries after all previous queries (queries 1 to $i-1$) failed to generate a confirm for $2 \leq i \leq L$. Consider the probability that a confirm is obtained upon the i_{th} query. For a given set of $P_1, P_2, \ldots, P_L$, as $\hat{\ell}$ is picked at random, the probability is $(1-P_1)\frac{L-1}{L}(1-P_2)\frac{L-2}{L-1}\ldots(1-P_i)\frac{1}{L-i} = \frac{1}{L}\prod_{j=1}^{i}(1-P_j)$. Therefore, the probability that the algorithm can get a confirm is $\sum_{i=1}^{L}[\frac{1}{L}\prod_{j=1}^{i}(1-P_j)]$.

The event that exactly i queries are made occurs when a confirm was obtained upon the i_{th} query, or when all first i queries failed to obtain the confirm and the algorithm decides to stop making queries. The probability is thus $[\prod_{j=1}^{i}(1-P_j)][\frac{1}{L}+\frac{L-i}{L}P_{i+1}]$. Note that P_{L+1} is not previously defined. However, as $\frac{L-i}{L} = 0$ when $i = L$, for convenience, we can use the same expression for all $1 \leq i \leq L$ for any arbitrary value of P_{L+1}. To derive the lower bound, we therefore need to solve the problem of minimizing $\sum_{i=1}^{L} i[\prod_{j=1}^{i}(1-P_j)][\frac{1}{L}+\frac{L-i}{L}P_{i+1}]$ subject to constraints that $\sum_{i=1}^{L}[\frac{1}{L}\prod_{j=1}^{i}(1-P_j)] = \epsilon - \frac{1}{2^V}$ and $0 \leq P_i \leq 1$. To solve the problem, we let $\eta_i =$

$\prod_{j=1}^{i}(1-P_j)$ and note that $P_{i+1} = 1 - \frac{\eta_{i+1}}{\eta_i}$. Therefore,

$$\begin{aligned}
& \sum_{i=1}^{L} i[\prod_{j=1}^{i}(1-P_j)][\frac{1}{L} + \frac{L-i}{L}P_{i+1}] \\
= & \sum_{i=1}^{L} i\eta_i[\frac{1}{L} + \frac{L-i}{L}(1-\frac{\eta_{i+1}}{\eta_i})] \\
= & \frac{1}{L}[\sum_{i=1}^{L}(L-i+1)\eta_i i - \sum_{i=1}^{L-1}(L-i)\eta_{i+1} i] \\
= & \frac{1}{L}[\sum_{i=1}^{L}(L-i+1)\eta_i].
\end{aligned}$$

We therefore consider a new problem of minimizing $\frac{1}{L}[\sum_{i=1}^{L}(L-i+1)\eta_i]$ subject to constraints that $\sum_{i=1}^{L}\eta_i = L(\epsilon - \frac{1}{2^V})$, $0 \le \eta_i \le 1$, $\eta_{i+1} \le \eta_i$. The optimal objective value for the newly defined problem must be no more than that of the original problem, because any valid assignment of $\{P_i\}_i$ gives a valid assignment of $\{\eta_i\}_i$. To achieve the optimal of the new problem, note that if $i < j$, the coefficient of η_i is more than that of η_j in the objective function, therefore, to minimize the objective function, we should reduce η_i and increase η_j. Considering that $\{\eta_i\}_i$ is nondecreasing, the optimal is achieved when all η_i are set to the same value $(\epsilon - \frac{1}{2^V})$, and the optimal value is $\frac{(\epsilon - \frac{1}{2^V})(L+1)}{2}$. ■

We also have the following lemma, the proof of which is given in the accompanying technical report [15].

Lemma 3.3: Suppose an algorithm makes hash queries to β index sets on average to solve the puzzle. Let $U(\beta)$ be the average number of unique indices in the index sets selected by the algorithm. A constant $\delta \in [0,1]$ exists and satisfies

$$U(\beta) \ge \frac{(1-\delta)n(1-e^{-Lk/n})\beta}{L}.$$

Note that the lemma is trivially true if $\delta = 1$; however, to make a useful bound, δ should be as small as possible. We show in the accompanying technical report [15] that with practical puzzle parameters, δ can be close to 0.

Suppose $\mathcal{A}$ has an advantage of σ in solving the puzzle when receiving $\omega(\mathcal{A})$ bits on average. Based on Theorem 3.1, $\mathcal{B}_{\mathcal{A}}$ has an advantage of no less than $\sigma - \frac{q_{\mathsf{hash}}}{2^V}$ while receiving no more than $\omega(\mathcal{A}) + \frac{Lkq_{\mathsf{hash}}}{2^V} + V$ bits on average. Based on Theorem 3.2, to achieve an advantage of at least $\sigma - \frac{q_{\mathsf{hash}}}{2^V}$, the optimal algorithm $\mathcal{B}$ must make at least $\frac{(\sigma - \frac{q_{\mathsf{hash}}+1}{2^V})(L+1)}{2}$ informed hash queries. Based on Lemma 3.3, also considering that $\mathcal{B}$ needs to receive only $k - V + 1$ bits per index set, $\mathcal{B}$ receives at least $U(\frac{(\sigma - \frac{q_{\mathsf{hash}}+1}{2^V})(L+1)}{2}) - L(V-1)$ bits on average. Therefore,

Theorem 3.4: Suppose $\mathcal{A}$ solves the puzzle with probability no less than σ. Let $\omega(\mathcal{A})$ denote the average number of received bits. We have

$$\begin{aligned}
\omega(\mathcal{A}) \ \ge \ & \frac{(1-\delta)n(1-e^{-Lk/n})(\sigma - \frac{q_{\mathsf{hash}}+1}{2^V})(L+1)}{2L} \\
& -L(V-1) - \frac{Lkq_{\mathsf{hash}}}{2^V} - V
\end{aligned}$$

where q_{hash}, V, and δ are constants determined by the puzzle parameters.

B. Multiple Adversaries with Multiple Puzzles

For the case where multiple adversaries are required to solve multiple puzzles, the proof uses the same idea as the single adversary case. Due to the limit of space, the complete proof is provided in the accompanying technical report [15]. Basically, we extend Ω to handle multiple adversaries, where Ω gives correct answer to a hash query from an adversary only if the number of bits the adversary received for the index set is greater than $k - V$, *regardless of the number of bits other adversaries received*. With similar arguments as the single adversary case, we consider the average number of bits received among the adversaries, and establish the relationship between the algorithm performance when interacting with Ω and with the real oracles. We then obtain the average number of informed queries the adversaries must make to achieve certain advantages when interacting with Ω. After solving the related optimization problems, we prove that

Theorem 3.5: Suppose A adversaries are challenged with P puzzles. Suppose $\mathcal{A}$ solves the puzzle with probability no less than σ and let $\omega(\mathcal{A})$ denote the average number of received bits. We have

$$\begin{aligned}
\omega(\mathcal{A}) \ \ge \ & \frac{(1-\delta)nP(\sigma - \frac{q_{\mathsf{hash}}+1}{2^V})(L+1)(1-e^{-q_{\mathsf{hash}}k/n})}{2q_{\mathsf{hash}}} \\
& -PL(V-1) - \frac{PLkAq_{\mathsf{hash}}}{2^V} - VP
\end{aligned}$$

where q_{hash}, V, and δ are constants determined by the puzzle parameters.

C. Achieving the Bound

We note that there exists a simple strategy the adversaries may adopt to be compared with the bound. In this strategy, when challenged with the puzzles, the adversaries flip a coin and decide whether to attempt to solve the puzzles. They attempt with probability σ; otherwise they simply ignore the puzzles. If they decide to solve the puzzles, the adversities select $\frac{P(L+1)}{2q_{\mathsf{hash}}}$ members, and let each of them get the entire content. Each of the chosen adversaries makes q_{hash} hash queries allowed for them. For each puzzle, the adversaries make hash queries for the index sets one by one until a $\mathsf{confirm}$ is obtained.

We now analyze the performance of this strategy. We argue that the adversaries can solve the puzzles with probability close to 1 if they decide to attempt, hence their advantage is σ. Note that to get a $\mathsf{confirm}$ for puzzle according to this strategy, the number of hash queries follows a uniform distribution in $[1, L]$ and is independent of other puzzles. The total number of hash queries is a random variable with mean $\frac{P(L+1)}{2}$. As the

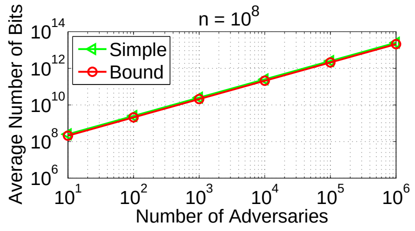

Fig. 1. Average number of bits needed by the simple strategy and the bound.

number of puzzles increases, the distribution of this variable approaches a Gaussian distribution centered around the mean with decreasing variance. Therefore, if the adversaries can make $\frac{P(L+1)}{2}$ hash queries, the probability that they can solve the puzzles asymptotically approaches 1. Note that this is possible because there are $\frac{P(L+1)}{2q_{\mathsf{hash}}}$ selected adversaries, each making q_{hash} queries.

According this strategy, the average number of bits downloaded is $\frac{\sigma n P(L+1)}{2q_{\mathsf{hash}}}$. Comparing to the bound in Theorem 3.5, it is a small fraction from $\frac{(1-\delta)nP(\sigma-\frac{q_{\mathsf{hash}}+1}{2^V})(L+1)(1-e^{-q_{\mathsf{hash}}k/n})}{2q_{\mathsf{hash}}}$ which is the dominant term, provided that δ, $\frac{q_{\mathsf{hash}}+1}{2^V}$, and $e^{-q_{\mathsf{hash}}k/n}$ are all small. We discuss in the accompanying technical report [15] that these conditions are true for a wide range of puzzle parameters. Therefore, this strategy approaches the bound asymptotically within a small fraction.

For instance, when $n = 10^8$, possible parameters are: $k = 10^4$, $L = 2 \times 10^3$, $z = 10$, and a θ such that $q_{\mathsf{hash}} = 4 \times 10^4$. Under such conditions, it is possible to set $\delta = 0.1$ and $V = 60$, and it can be verified that both $e^{-q_{\mathsf{hash}}k/n}$ and $\frac{q_{\mathsf{hash}}+1}{2^V}$ are small. Fig. 1 shows the bound and the simple strategy when $\sigma = 1$ for various number of adversaries under these parameters, where we can see that the difference is small. Due to the limit of space, we discuss practical choices of the puzzle parameters in [15].

IV. Related Work

Using puzzles has been proposed (e.g., in [8], [10], [1], [7], [6]) to defend against email spamming or denial of service attacks. In these schemes, the clients are required to spend time to solve puzzles before getting access to the service. The purpose of the bandwidth puzzle is to verify whether the claimed content transactions took place, where the ability to solve the puzzles is tied to the amount contents actually downloaded. As such, the construction of the bandwidth puzzle is different from existing puzzles.

Proofs of data possession (PDP) (e.g., [2], [9], [3]) and Proofs of retrievability (POR) (e.g., [11], [5], [13]) have been proposed to allow a client to verify whether the data has been modified in a remote store. As discussed in [12], the key differences between PDP/POR schemes and the bandwidth puzzle include the following. First, PDP/POR assumes a single verifier and prover, while the bandwidth puzzle considers one verifier with many potentially colluding provers. Second, the bandwidth puzzle has low computational cost at the verifier, which is desirable in the case when one verifier has to handle many provers, while the existing PDP/POR schemes may incur heavy computational cost at the verifier. The proof techniques for PDP/POR schemes are also different from the techniques used in this paper, because collusion is not considered in existing PDP/POR schemes.

In our earlier work [12], an upper bound was given on the expected number puzzles that can be solved if the adversaries are allowed and a certain number of hash queries and content queries. In this work, we remove assumption on the maximum number of content queries. With less assumptions, our proof is less restrictive and applies to more general cases. The new problem is different from the problem studied in [12], and new techniques are developed to establish the bound. Note that although the adversaries are allowed to download as many bits as they wish, they prefer to employ an intelligent algorithm to minimize the number of downloaded bits. The new bound guarantees that, if the adversaries wish to have a certain advantage in solving the puzzle, there exist a lower bound on the average number of bits they have to download, regardless of the algorithm they adopt.

V. Conclusions

In this paper, we prove a new bound on the performance of the bandwidth puzzle which has been proposed to defend against colluding adversaries in p2p content distribution networks. Our proof is based on reduction, and gives the lower bound of the average number of downloaded bits to achieve a certain advantage by the adversaries. The bound is asymptotically tight in the sense that it is a small fraction away from the average number of bits downloaded when following a simple strategy. The new bound is a significant improvement over the existing bound which is derived under more restrictive conditions and is much looser. The improved bound can be used to guide the choice of puzzle parameters to improve the performance of practical systems.

References

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

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

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

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

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

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

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

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

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

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

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

[12] M. Reiter, V. Sekar, C. Spensky, and Z. Zhang. Making contribution-aware p2p systems robust to collusion attacks using bandwidth puzzles. In *Proc. of ICISS*, 2009.

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

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

[15] Z. Zhang. A new bound on the performance of the bandwidth puzzle. In *arXiv:1102.3745v1*, 2011.