A New Bound on the Performance of the Bandwidth Puzzle

Zhenghao Zhang, *Member, IEEE*

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 content 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 content 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 with practical puzzle parameters. The new bound gives better security guarantees than the existing bound, and can be used to guide the choices of puzzle parameters for practical systems.

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 [**?**], [**?**], [**?**]. Without a robust incentive mechanism, peers may choose not to upload content for other peers, causing the entire system to fail. In many applications, a peer's contribution is measured by the number of bits it uploads 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

Z. Zhang is with the Computer Science Department, Florida State University, Tallahassee, FL 32306. E-mail: zzhang@cs.fsu.edu.

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 [**?**]. 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 if 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 content. To illustrate the basic idea of the puzzle, we may 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 Alice receives the puzzle, she can solve the puzzle 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 can be implemented in practical systems to thwart collusions. It is most suited for real time video broadcast applications, where the verifier can naturally reside in the source node of the video, and the puzzles can be generated based on the unique content currently being broadcast [**?**]. It is desirable for the content to be fresh, because otherwise, the adversaries may use out-of-band channels to "smuggle" the content in order to game the system. For example, if the puzzles are based on static content, Alice may burn the content she has into DVDs and give them to all her friends, who may be living in the same building. This will enable Alice's friends to solve the puzzles, and trick the system into believing that Alice contributed a huge amount of network bandwidth to the system.

The construction of the bandwidth puzzle is simple and is based only on hash functions and pseudorandom functions. In [**?**], 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, for practical puzzle parameters, there exists a strategy that approaches 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 **??** describes the construction of the puzzle. Section **??** gives the proof of the new bound. Section **??** discusses practical puzzle parameters and shows how a simple strategy approaches the bound. Section **??** discusses related works. Section **??** concludes the paper.

II. The Construction

In this section, we describe the construction of the puzzle. The puzzle construction is largely the same as [**?**] except one difference: allowing repeated indices in one sequence (the definition of sequence 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 **??**.

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 *sequence* is defined as k ordered indices chosen from the n indices. Each sequence defines a string denoted as str, called the *true string* of this sequence, 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 sequence. To construct a puzzle, the verifier needs L sequences denoted as $I_1, \ldots, I_L$, where a sequence is obtained by randomly choosing the indices, allowing repeat. The verifier randomly chooses one sequence among the L sequences, denoted as $I_{\hat{\ell}}$, called the *answer sequence*. 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 sequences and $\hat{h}$. When challenged with a puzzle, the prover should prove that it knows which sequence 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.

n	The number of bits in the content
k	The number of indices in a sequence
L	The number of sequences 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

The strengths of the puzzle are: 1) a prover must have the content, otherwise it cannot get the true strings of the sequences, and 2) even if the prover has the content, it still has to spend time and try different sequences until it finds a sequence with the same hash as the hint, refereed to as a confirm event, because the hash function is one-way. In practice, the verifier need not generate all sequences; it need only generate and find the hash of the answer sequence. The verifier should not send the L sequences 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 sequences 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 sequences in the puzzle are $I_1 = (6; 4; 8; 1)$, $I_2 = (2; 3; 7; 4)$, and $I_3 = (3; 4; 6; 4)$. 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 sequences, 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 sequence.

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.

Pseudorandom functions are used to generate the sequences. 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: $\{f_K^1 : \{1, \ldots, L\} \rightarrow \{0, 1\}^\kappa\}$ and $\{f_K^2 : \{1, \ldots, k\} \rightarrow \{1, \ldots, n\}\}$.

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:

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 missing bits
Υ	The constant for the *Uniqueness Constraint*
δ	A constant representing the deviation from the mean

TABLE 2

LIST OF NOTATIONS IN THE PROOF

$\{0,1\}^{\kappa} \times \{1,\ldots,L\} \times \{0,1\}^{k} \rightarrow \{0,1\}^{\kappa}$. To prove the security of the puzzle, hash is modeled as a random oracle [**?**]. 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 sequences. 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 sequence. With K_1 and $\hat{\ell}$, it generates $K_2^{\hat{\ell}} \leftarrow f^1_{K_1}(\hat{\ell})$. $K_2^{\hat{\ell}}$ is used as the key for $f^2_{K_2}$ to generate the indices in the answer sequence: $I_{\hat{\ell}} = \{f^2_{K_2^{\hat{\ell}}}(1)\ldots f^2_{K_2^{\hat{\ell}}}(k)\}$. The verifier then finds $str_{\hat{\ell}}$, and 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 sequences in the same way as the verifier generates the answer sequence, and can compare the hash of the sequences 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.

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 **??**.

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 $\mathsf{content}$. The adversary is given access to hash, ans, and $\mathsf{content}$. 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 $\mathsf{content}$ and ans. We refer a query to $\mathsf{content}$ as a *content query* and a query to hash a *hash query*. We use $\mathcal{G}$ to denote the algorithm adopted by the adversary. We use $\omega()$ to 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.

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 $\mathsf{content}$ *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 sequence in the first attempt of hash queries, only k content queries are needed. However, the adversary may be lucky once but may unlikely to be always lucky. Therefore, if challenged with a large number of puzzles, the average number of queries it makes to $\mathsf{content}$ 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 $\mathcal{G}$ the adversaries may adopt, we construct an algorithm $\mathcal{B}_{\mathcal{G}}$ that employs $\mathcal{G}$ and implements oracle queries for $\mathcal{G}$. $\mathcal{B}_{\mathcal{G}}$ terminates when $\mathcal{G}$ terminates, and returns what $\mathcal{G}$ returns. When $\mathcal{G}$ makes a query, $\mathcal{B}_{\mathcal{G}}$ replies as follows:

Algorithm 1 $\mathcal{B}_{\mathcal{G}}$ answers oracle queries for $\mathcal{G}$

```
1: When $\mathcal{G}$ makes a query to content, $\mathcal{B}_{\mathcal{G}}$ makes the same content query to $\Omega$ and **returns** the result to $\mathcal{G}$.
2: When $\mathcal{G}$ makes a query to ans, $\mathcal{B}_{\mathcal{G}}$ makes the same query to ans and **returns** the result to $\mathcal{G}$.
3: When $\mathcal{G}$ makes a query to hash for $I_\ell$:
    1) $\mathcal{B}_{\mathcal{G}}$ checks if $\mathcal{G}$ has made exactly the same query before. If yes, it **returns** the last answer.
    2) $\mathcal{B}_{\mathcal{G}}$ checks if $\mathcal{G}$ has made content queries for more than $k - V$ bits in $I_\ell$. If not, it **returns** a random string.
    3) If $\mathcal{B}_{\mathcal{G}}$ has not made a hash query for $I_\ell$ before, $\mathcal{B}_{\mathcal{G}}$ makes a hash query to $\Omega$. Depending on if confirm is obtained upon this query, $\mathcal{B}_{\mathcal{G}}$ knows if $I_\ell$ is the answer sequence. If $I_\ell$ is the answer sequence, $\mathcal{B}_{\mathcal{G}}$ sends content queries to $\Omega$ to get the remaining bits in $I_\ell$.
    4) If $I_\ell$ is not the answer sequence, $\mathcal{B}_{\mathcal{G}}$ **returns** a random string.
    5) If the string $\mathcal{G}$ submitted is the not true string of $I_\ell$, $\mathcal{B}_{\mathcal{G}}$ **returns** a random string.
    6) $\mathcal{B}_{\mathcal{G}}$ **returns** the hash of $I_\ell$.
```

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

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

and

$$\omega[\mathcal{B}_{\mathcal{G}}] \leq \omega[\mathcal{G}] + \frac{Lkq_{\mathsf{hash}}}{2^V} + V - 1.$$

Proof: In our construction, $\mathcal{B}_{\mathcal{G}}$ employs $\mathcal{G}$, and answers oracle queries for $\mathcal{G}$. Denote the random process of $\mathcal{G}$ when it is interacting directly with content, hash and ans as W, and denote the random process of $\mathcal{G}$ when it is interacting with the oracles implemented by $\mathcal{B}_{\mathcal{G}}$ 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 $\mathcal{G}$ makes a query to $\mathsf{content}$ or ans, $\mathcal{B}_{\mathcal{G}}$ simply gives the query result, therefore the only case needs to be considered is when $\mathcal{G}$ makes a query to hash. When $\mathcal{G}$ makes a query for I_ℓ to hash,

- If there are still no less than V unknown bits in I_ℓ, $\mathcal{B}_{\mathcal{G}}$ 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 $\mathsf{confirm}$, and this will have same effect on the progress of the algorithm statistically as when $\mathcal{G}$ is making a query to hash. However, if $\ell = \hat{\ell}$, it could happen that $\mathcal{G}$ 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 $\mathcal{B}_{\mathcal{G}}$ has made enough content queries for I_ℓ, $\mathcal{B}_{\mathcal{G}}$ checks if it has made hash query for I_ℓ before. If not, $\mathcal{B}_{\mathcal{G}}$ makes the hash query, and if a $\mathsf{confirm}$ is obtained, $\mathcal{B}_{\mathcal{G}}$ knows that I_ℓ is the answer sequence and gets the possible remaining bits in I_ℓ; otherwise $\mathcal{B}_{\mathcal{G}}$ knows that I_ℓ is not the answer sequence. If I_ℓ is not the answer sequence, $\mathcal{B}_{\mathcal{G}}$ will simply return a random string, which will have the same effect statistically on the progress of $\mathcal{G}$ as when $\mathcal{G}$ is interacting with hash. If I_ℓ is the answer sequence, $\mathcal{B}_{\mathcal{G}}$ checks if $\mathcal{G}$ is submitting the true string, and returns a random string if not and the true hash otherwise. This, clearly, also has the same effect statistically on the progress of $\mathcal{G}$ as when $\mathcal{G}$ is interacting with hash.

From the above discussion, we can see that $\mathsf{P}\left[C_{\mathcal{B}_{\mathcal{G}}}\right]$ is no less than $\mathsf{P}\left[C_{\mathcal{G}}\right]$ minus the probability of the exception. Therefore, the first half of the theorem is proved. We can also see that if the exception occurs, $\mathcal{B}_{\mathcal{G}}$ makes at most Lk more content queries than $\mathcal{G}$. If the exception does not occur, $\mathcal{B}_{\mathcal{G}}$ receives at most $V-1$ more bits than $\mathcal{G}$ it encapsulates, and therefore at most $V-1$ bits more than $\mathcal{G}$ on average when $\mathcal{G}$ is interacting directly with $\mathsf{content}$, hash and ans. ■

Theorem **??** 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 $\mathsf{confirm}$.

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 $\mathsf{correct}$ denote the event that $\mathcal{B}^*$ returns the correct answer. We have

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

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 sequence, since ans is collision-resistant. If a $\mathsf{confirm}$ was not obtained, the answer sequence is missing no less than V bits, since otherwise an optimal algorithm should make a query which will result in a $\mathsf{confirm}$. Therefore, the probability that the algorithm can obtain the true string of the answer sequence 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 $\epsilon - \frac{1}{2^V}$ in getting $\mathsf{confirm}$.

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 $\mathsf{confirm}$ for $2 \leq i \leq L$. Consider the probability that a $\mathsf{confirm}$ is obtained upon the i_{th} query. For a given set of $P_1, P_2, \ldots, P_L$, because $\hat{\ell}$ 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 $\mathsf{confirm}$ 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 $\mathsf{confirm}$ was obtained upon the i_{th} query, or when all first i queries failed to obtain the $\mathsf{confirm}$ 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 $1\le i\le L$ 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 the constraints that $\sum_{i=1}^{L}[\frac{1}{L}\prod_{j=1}^{i}(1-P_j)]=\epsilon-\frac{1}{2^V}$ and $0\le P_i\le 1$.

To solve the problem, we let $\eta_i=\prod_{j=1}^{i}(1-P_j)$. 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 the constraints that $\sum_{i=1}^{L}\eta_i=L(\epsilon-\frac{1}{2^V})$, $0\le\eta_i\le 1$, and $\eta_{i+1}\le\eta_i$. The optimal 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 value of the new problem, note that if $i<j$, the coefficient of η_i is more than 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 $\frac{(\epsilon-\frac{1}{2^V})(L+1)}{2}$. ■

Based on Theorem **??**, any algorithm with an advantage of ϵ must make no less than a certain number of informed hash queries to Ω on average. We next derive the number of unique indices in a given number sequences. We first need the following lemma.

Lemma 3.3: Suppose c indices are randomly picked among n indices, with repeat. Let Y be the random number denoting the number of unique indices among the c indices. For any constant $0<\delta<1$, let

$\mu = n(1-\delta)(1-e^{-\frac{c}{n}})$ and $\eta = \frac{c - n\ln\frac{n}{n-\mu}}{n\sqrt{(\frac{1}{n-\mu}-\frac{1}{n})}}$. We have

$$\mathsf{P}\left[Y < \mu\right] < \frac{e^{-\eta^2/2}}{\sqrt{2\pi}\eta}$$

when $n \to \infty$ and $c \to \infty$.

Proof: Consider the process when indices are randomly taken from n indices. Let Z_i be the random number denoting the number of samples needed to get the i_{th} unique index. Clearly, $\mathsf{P}\left[Z_1 = 1\right] = 1$. In general, Z_i follows the geometric distribution, i.e.,

$$\mathsf{P}\left[Z_i = j\right] = (\frac{i-1}{n})^{j-1}\frac{n-i+1}{n}.$$

Let $p_i = \frac{n-i+1}{n}$, we have $\mathsf{E}\left[Z_i\right] = \frac{1}{p_i}$, and $\mathsf{Var}\left[Z_i\right] = \frac{1-p_i}{p_i^2}$. Also, $\{Z_i\}_i$ are independent of each other. Define $S_\mu = \sum_{i=1}^{\mu} Z_i$ and note that $\mathsf{P}\left[Y < \mu\right] = \mathsf{P}\left[S_\mu > c\right]$. Therefore, we will focus on finding $\mathsf{P}\left[S_\mu > c\right]$.

Define $Z_i' = Z_i - \mathsf{E}\left[Z_i\right]$. Let $S_\mu' = \sum_{i=1}^{\mu} Z_i'$, clearly, $\mathsf{P}\left[S_\mu > c\right] = \mathsf{P}\left[S_\mu' > c - \sum_{i=1}^{\mu}\frac{1}{p_i}\right]$. As $\{Z_i'\}_i$ are independent random variables with zero mean, due to the Central Limit Theorem, S_μ' follows the Gaussian distribution with zero mean and variance $\sum_{i=1}^{\mu}\frac{1-p_i}{p_i^2}$ as $\mu \to \infty$.

We note that $c > \sum_{i=1}^{\mu}\frac{1}{p_i}$. This is because

$$\sum_{i=1}^{\mu}\frac{1}{p_i} = \frac{n}{n} + \frac{n}{n-1} + ... + \frac{n}{n-\mu+1} < n\ln\frac{n}{n-\mu},$$

while

$$\begin{aligned} & c > n\ln\frac{n}{n-\mu} \\ \Leftrightarrow\quad & e^{\frac{c}{n}} > \frac{1}{1-(1-\delta)(1-e^{-\frac{c}{n}})} \\ \Leftrightarrow\quad & e^{\frac{c}{n}} - (1-\delta)e^{\frac{c}{n}} + (1-\delta) > 1 \\ \Leftrightarrow\quad & \delta e^{\frac{c}{n}} - \delta > 0 \end{aligned}$$

which is true because $e^{\frac{c}{n}} > 1$. Therefore,

$$\mathsf{P}\left[S_\mu' > c - \sum_{i=1}^{\mu}\frac{1}{p_i}\right] \leq Q(\frac{c - \sum_{i=1}^{\mu}\frac{1}{p_i}}{\sqrt{\sum_{i=1}^{\mu}\frac{1-p_i}{p_i^2}}}),$$

where $Q()$ is the Gaussian Error Integral. To simplify the result, note that

$$\begin{aligned}\sum_{i=1}^{\mu}\frac{1-p_i}{p_i^2} &= \sum_{i'=0}^{\mu-1}\frac{1-\frac{n-i'}{n}}{(\frac{n-i'}{n})^2}=\sum_{i'=0}^{\mu-1}\frac{ni'}{(n-i')^2}\\ &= n\sum_{i'=0}^{\mu-1}[\frac{n}{(n-i')^2}-\frac{n-i'}{(n-i')^2}]\\ &< n^2\sum_{i'=0}^{\mu-1}\frac{1}{(n-i')^2}\\ &< n^2(\frac{1}{n-\mu}-\frac{1}{n})\end{aligned}$$

Applying the bounds, we have

$$\mathsf{P}\left[Y<\mu\right]<Q(\frac{c-n\ln\frac{n}{n-\mu}}{n\sqrt{(\frac{1}{n-\mu}-\frac{1}{n})}})$$

The proof completes since $Q(x)\le\frac{e^{-x^2/2}}{\sqrt{2\pi}x}$ **[?]**. ■

Note that according to the well-known Coupon Collector problem, $n(1-e^{-\frac{c}{n}})$ is actually the average number of unique indices among c indices, and δ determines how far μ deviates from this value. This lemma establishes the bound for the probability that the number of unique indices is less than $1-\delta$ fraction of the average.

Let Y_s^J denote the minimum number of unique indices in s sequences among all possible choices of s sequences picked from J sequences where $s\ge 1$. Let $\mu_s=n(1-\delta)(1-e^{-sk/n})$ and $\eta_s=\frac{sk-n\ln\frac{n}{n-\mu_s}}{n\sqrt{(\frac{1}{n-\mu_s}-\frac{1}{n})}}$. Based on Lemma **??**, we have

$$\mathsf{P}\left[Y_s^J<\mu_s\right]<\frac{e^{-\eta_s^2/2}}{\sqrt{2\pi}\eta_s}J^s$$

as $n\to\infty$ and $k\to\infty$. This is because event $Y_s^J<\mu_s$ happens if one combination of s sequences have less than μ_s unique indices, the probability of which is bounded by Lemma **??**, and the total number of combinations to pick s sequences is less than J^s. For a given δ, we introduce a constraint called the *Uniqueness Constraint* and a constant Υ:

- A set of J sequences satisfy the *Uniqueness Constraint* if $Y_s^J\ge\mu_s$ for any $s\le J$.
- In our proof, we specifically choose a Υ such that all sequences in the puzzle (or in the puzzles in the multiple adversary case) satisfy the *Uniqueness Constraint* with probability no less than $1-\Upsilon$.

Clearly, Υ depends on δ. We discuss in Section **??** and show that for practical puzzles, when $J\le 10^{12}$, δ can be as small as 0.1 while Υ can be as small as 10^{-10}.

To establish the bound, the next lemma is also needed.

Lemma 3.4: Consider a linear programing problem of maximizing $\sum_{i=0}^{L} P_i e^{-id}$ subject to the constraints that $\sum_{i=0}^{L} P_i i = \beta$, $\sum_{i=0}^{L} P_i = \gamma$, $P_i \geq 0$, where $0 \leq d$, $0 \leq \gamma \leq 1$ and $0 \leq \beta \leq \gamma L$. Denote the optimal value of the objective function as $F_L(\beta, \gamma)$, we have

$$F_L(\beta, \gamma) = \gamma - \frac{(1 - e^{-Ld})}{L}\beta$$

That is, to achieve the optimal value is to let $P_0 = \gamma - \frac{\beta}{L}$ and $P_L = \frac{\beta}{L}$, while letting $P_i = 0$ for $0 < i < L$.

Proof: We will use induction on L. To begin with, consider when $L = 1$. In this case, due to the constraints, P_0 and P_1 can be uniquely determined as $P_0 = \gamma - \beta$ and $P_1 = \beta$. Therefore,

$$F_1(\beta, \gamma) = \gamma - \beta + \beta e^{-d},$$

and the lemma is true. Suppose the lemma is true till $L = j$. To get $F_{j+1}(\beta, \gamma)$, suppose $P_{j+1} = \gamma - \gamma'$, where $\gamma' \leq \gamma$. Given this, let

$$\beta' = \sum_{i=0}^{j} P_i i = \beta - (\gamma - \gamma')(j + 1),$$

and therefore,

$$\begin{aligned} F_{j+1}(\beta, \gamma) &= F_j(\beta', \gamma') + (\gamma - \gamma')e^{-(j+1)d} \\ &= \gamma' - \frac{1 - e^{-jd}}{j}\beta' + (\gamma - \gamma')e^{-(j+1)d} \\ &= \gamma'[1 - \frac{(1 - e^{-jd})(j + 1)}{j} - e^{-(j+1)d}] - \frac{1 - e^{-jd}}{j}[\beta - \gamma(j + 1)] + \gamma e^{-(j+1)d} \end{aligned}$$

Regarding γ' as a variable, its coefficient is

$$1 - \frac{(1 - e^{-jd})(j + 1)}{j} - e^{-(j+1)d},$$

which is no more than 0. To see this, consider function

$$f(x) = 1 - \frac{(1 - e^{-jx})(j + 1)}{j} - e^{-(j+1)x}.$$

Note that $f(0) = 0$, and $f'(x) < 0$ when $x \geq 0$. Therefore, to maximize the objective function, γ' should be as small as possible. Note that $jP_j = \beta - (j + 1)\gamma + (j + 1)\gamma'$ and $P_0 + P_j = \gamma'$. Therefore, $P_0 = \frac{(j+1)\gamma - \beta - \gamma'}{j}$. Since $P_0 \leq \gamma'$, we have $\gamma' \geq \gamma - \frac{\beta}{j+1}$. Therefore, $F_{j+1}(\beta, \gamma) = \gamma - \frac{\beta}{j+1} + \frac{\beta}{j+1}e^{-(j+1)d}$.

■

We next wish to bound from below the expected number of unique indices when β sequences form a puzzle are picked in expectation, denoted as $U(\beta)$. We know that with probability no more than Υ, the sequences do not satisfy the *Uniqueness Constraint*, in which case we simply use a bound 0. Otherwise, let P_i denote the probability that i sequences are picked where $0 \leq i \leq L$, the expected number of unique indices should be no less than

$$\sum_{i=0}^{L}(1-\delta)n(1-e^{-ik/n})P_i.$$

Therefore, to derive the lower bound is to maximize $\sum_{i=0}^{L} P_i e^{-ik/n}$ subject to the constraints that $\sum_{i=0}^{L} P_i i = \beta$, $\sum_{i=0}^{L} P_i = 1$, and $P_i \geq 0$. Based on Lemma **??**, we immediately have

Lemma 3.5: If the average number of picked sequences is β, then

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

as $n \to \infty$ and $k \to \infty$, where δ and Υ are constants determined by the puzzle parameters.

We may finally assemble the parts together and obtain the bound. Suppose algorithm $\mathcal{G}$ has an advantage of σ in solving the puzzle when receiving $\omega(\mathcal{G})$ bits on average. Based on Theorem **??**, $\mathcal{B}_{\mathcal{G}}$ has an advantage of no less than $\sigma - \frac{q_{\mathsf{hash}}}{2^V}$ while receiving no more than $\omega(\mathcal{G}) + \frac{Lkq_{\mathsf{hash}}}{2^V} + V - 1$ bits on average. Based on Theorem **??**, 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 **??**, also considering that $\mathcal{B}^*$ needs to receive only $k - V + 1$ bits per sequence, $\mathcal{B}^*$ receives at least $U(\frac{(\sigma - \frac{q_{\mathsf{hash}}+1}{2^V})(L+1)}{2}) - L(V-1)$ bits on average. Therefore,

Theorem 3.6: Suppose $\mathcal{G}$ solves the puzzle with probability no less than σ. We have

$$\omega(\mathcal{G}) \geq \frac{(1-\Upsilon)(1-\delta)n(1-e^{-Lk/n})(\sigma - \frac{q_{\mathsf{hash}}+1}{2^V})(L+1)}{2L} - (L+1)(V-1) - \frac{Lkq_{\mathsf{hash}}}{2^V}$$

as $n \to \infty$ and $k \to \infty$, where q_{hash}, Υ, V, and δ are constants determined by the puzzle parameters.

B. Multiple Adversaries with Multiple Puzzles

We next consider the more complicated case when multiple adversaries are required to solve multiple puzzles. Suppose there are A adversaries, and the number of puzzles they attempt to solve is P. Note that P is greater than A when $z > 1$.

1) Proof Sketch: The proof uses the same idea as the single adversary case. Basically, we extend Ω to handle multiple adversaries, where Ω gives the correct answer to a hash query from an adversary only if the number of bits the adversary received for the sequence is greater than $k - V$, *regardless of the number of bits other adversaries received*. With similar arguments as the single adversary case, we can establish the relationship between the algorithm performance when interacting with Ω and with the real oracles. We also obtain the average number of informed queries the adversaries must make to achieve certain advantages when interacting with Ω. The bound is established after solving several optimization problems.

2) Proof Details: Suppose the adversaries run an algorithm $\mathcal{G}$ that solves the P puzzles with probability σ while receiving $\omega(\mathcal{G})$ bits on average. We wish to bound from below $\omega(\mathcal{G})$ for a given σ. We extend the definition of Ω and let it remember the content queries from each adversary. We use I_ℓ^p to denote sequence ℓ in puzzle p where $1 \leq p \leq P$. If an adversary makes a hash query for I_ℓ^p while this adversary has made content query for more than $k - V$ bits in I_ℓ^p, Ω replies with the hash of I_ℓ^p, otherwise, it returns $\emptyset$. In addition, if the adversaries have made more than L hash queries for a particular puzzle, Ω will not answer further hash queries for this puzzle.

Similar to the single puzzle case, given an algorithm $\mathcal{G}$ for solving the puzzles, we construct an algorithm $\mathcal{B}_\mathcal{G}$ employing $\mathcal{G}$ denoted as $\mathcal{B}_\mathcal{G}$. $\mathcal{B}_\mathcal{G}$ terminates when $\mathcal{G}$ terminates, and returns what $\mathcal{G}$ returns. Algorithm **??** describes how $\mathcal{B}_\mathcal{G}$ implements oracle queries for $\mathcal{G}$, which is very similar to the single adversary case.

Algorithm 2 $\mathcal{B}_\mathcal{G}$ answers oracle queries for $\mathcal{G}$

```
1: When $\mathcal{G}$ makes a query to content, $\mathcal{B}_\mathcal{G}$ makes the same content query to $\Omega$ and gives the result to $\mathcal{G}$.
2: When $\mathcal{G}$ makes a query to ans, $\mathcal{B}_\mathcal{G}$ makes the same query to ans and gives the result to $\mathcal{G}$.
3: When $\mathcal{G}$ makes a query for $I_\ell^p$ to hash at adversary $v$:
     1) $\mathcal{B}_\mathcal{G}$ checks if adversary $v$ has made exactly the same query before. If yes, it **returns** the last answer.
     2) $\mathcal{B}_\mathcal{G}$ checks if adversary $v$ has made content queries for more than $k - V$ bits in $I_\ell^p$, and if this is not true, it **returns** a random string.
     3) If $\mathcal{B}_\mathcal{G}$ has not made a hash query for $I_\ell^p$, it makes a hash query to $\Omega$. Depending on if confirm is obtained upon this query, $\mathcal{B}_\mathcal{G}$ knows if $I_\ell^p$ is the answer sequence of puzzle $p$. If $I_\ell^p$ is the answer sequence, $\mathcal{B}_\mathcal{G}$ sends content queries to $\Omega$ to get the remaining bits in $I_\ell^p$.
     4) If $I_\ell^p$ is not the answer sequence of puzzle $p$, $\mathcal{B}_\mathcal{G}$ **returns** a random string.
     5) If the string $\mathcal{G}$ submitted is the not true string of $I_\ell^p$, $\mathcal{B}_\mathcal{G}$ **returns** a random string.
     6) $\mathcal{B}_\mathcal{G}$ **returns** the hash of $I_\ell^p$.
```

With very similar arguments as in Theorem **??**, we have

Theorem 3.7: Let $C_\mathcal{G}$ be the event that $\mathcal{G}$ returns the correct answers when it is interacting directly with content, hash and ans. Let $C_{\mathcal{B}_\mathcal{G}}$ be the event that $\mathcal{B}_\mathcal{G}$ returns the correct answers, when it is interacting with Ω. Then,

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

and

$$\omega[\mathcal{B}_{\mathcal{G}}] \leq \omega[\mathcal{G}] + \frac{PLkAq_{\mathsf{hash}}}{2^V} + (V-1)P.$$

Let $\mathcal{B}^*$ denote the optimal algorithm for solving the puzzles when the algorithm is interacting with Ω. Note that if the probability that $\mathcal{B}^*$ solves all puzzles is no less than ϵ, the probability that an individual puzzle is solved is no less than ϵ. Based on Theorem **??**, if a puzzle is solved with probability no less than ϵ, the average number of informed hash queries made for this puzzle is no less than $\frac{(\epsilon-\frac{1}{2^V})(L+1)}{2}$. For P puzzles, we obtain the following theorem due to the linearity of expectation.

Theorem 3.8: If the probability that $\mathcal{B}^*$ solves all puzzles is no less than ϵ, on average, the number of informed hash queries is no less than $\frac{P(\epsilon-\frac{1}{2^V})(L+1)}{2}$.

Next, we wish to bound from below the number of unique indices in the involved sequences, whenever the adversaries collectively have to make T hash queries. Here we define the unique indices at adversary v as the total number of unique indices in the sequences that it makes hash queries for, and denote it as u_v. The total number of unique indices is defined as $\sum_{v=1}^{A} u_v$. The adversaries may assign hash queries intelligently, such that $\sum_{v=1}^{A} u_v$ is minimized. For example, if two sequences share a large number of indices, they should be assigned to the same adversary. Nevertheless, we have

Lemma 3.9: If the number of queries is T and if the involved sequences satisfy the *Uniqueness Constraint,*

$$\sum_{v=1}^{A} u_v \geq (1-\delta)n[t(1-e^{-q_{\mathsf{hash}}k/n}) + (1-e^{-(T-q_{\mathsf{hash}}t)k/n})]$$

as $n \to \infty$ and $k \to \infty$, where $\mid x \mid^-$ denotes the largest integer no more than x and $t = \mid T/q_{\mathsf{hash}} \mid^-$.

Proof: Suppose the number of hash queries made by adversary v is s_v. We have

$$\sum_{v=1}^{A} u_v \geq \sum_{v=1}^{A} (1-\delta)n(1-e^{-s_v k/n})$$

Therefore, to minimize $\sum_{v=1}^{A} u_v$ is to maximize $\sum_{v=1}^{A} e^{-s_v k/n}$ subject to the constraints that $\sum_{v=1}^{A} s_v = T$ and $0 \leq s_v \leq q_{\mathsf{hash}}$.

We claim that the optimal is achieved when s_i is set to be q_{hash} for $1 \leq i \leq \mid T/q_{\mathsf{hash}} \mid^-$, which we show by induction on the number of adversaries. First consider when $A = 2$. If $T \leq q_{\mathsf{hash}}$, we claim that

$\sum_{i=1}^{2} e^{-s_i k/n}$ is maximized when $s_1 = T$ and $s_2 = 0$, which is because for any valid s_1 and s_2,

$$\begin{aligned} & (1+e^{-Tk/n}) - (e^{-s_1 k/n} + e^{-s_2 k/n}) \\ = \; & (1-e^{-s_1 k/n})(1-e^{-s_2 k/n}) \\ \geq \; & 0. \end{aligned}$$

Similarly, if $q_{\mathsf{hash}} \leq T \leq 2q_{\mathsf{hash}}$, $\sum_{i=1}^{A} e^{-s_i k/n}$ is maximized when $s_1 = q_{\mathsf{hash}}$ and $s_2 = T - q_{\mathsf{hash}}$, which is because for any valid s_1 and s_2,

$$\begin{aligned} & (e^{-q_{\mathsf{hash}} k/n} + e^{-(T-q_{\mathsf{hash}})k/n}) - (e^{-s_1 k/n} + e^{-s_2 k/n}) \\ = \; & (1-e^{(-s_1+T-q_{\mathsf{hash}})k/n})(e^{(-T+q_{\mathsf{hash}})k/n} - e^{-s_2 k/n}) \\ \geq \; & 0. \end{aligned}$$

Therefore our claim is true for $A = 2$. Suppose our claim is true for $A = j$. For $A = j+1$, suppose in the optimal assignment, $s_{j+1} = 0$. Then, our claim is true based on the induction hypothesis. If in the optimal assignment, $s_{j+1} > 0$, we argue that in the optimal assignment, $s_i = q_{\mathsf{hash}}$ for all $1 \leq i \leq j$, therefore our claim is still true. This because if $s_i < q_{\mathsf{hash}}$ for some i, we can increase s_i while decreasing s_{j+1}. Using similar arguments as for the case when $A = 2$, this will increase the objective function, thus violating the fact that the assignment is optimal. ■

Similar to the single adversary case, if the average number of informed hash queries is β, we need to bound from below the average number of unique indices in the involved sequences, denoted as $U(\beta)$.

Lemma 3.10: Consider when the adversaries are given P puzzles. If the average number of informed hash queries is β, as $n \to \infty$ and $k \to \infty$,

$$U(\beta) \geq \frac{(1-\Upsilon)(1-\delta)n\beta(1-e^{-q_{\mathsf{hash}} k/n})}{q_{\mathsf{hash}}}$$

Proof: Similar to the arguments leading to Lemma **??**, the involved sequences do not satisfy the *Uniqueness Constraint* with probability no more than Υ, in which case we simply use a bound 0. In the rest we focus on the case when the *Uniqueness Constraint* is satisfied.

Denote the probability that there are i queries as P_i, where $0 \leq i \leq PL$. For notational simplicity, in the proof for this lemma, we let $d = k/n$. Based on Lemma **??**, we want to bound from below

$$\sum_{i=0}^{PL} P_i[(t_i - 1)(1-e^{-q_{\mathsf{hash}} d}) + (1-e^{-[i-q_{\mathsf{hash}}(t_i-1)]d})]$$

under the constraints that $\sum_{i=0}^{PL} P_i i = \beta$, $\sum_{i=0}^{PL} P_i = 1$, and $P_i \geq 0$, where t_i is an integer satisfying $(t_i - 1)q_{\mathsf{hash}} \leq i < t_i q_{\mathsf{hash}}$. To solve this problem, suppose C is the minimum integer satisfying $PL \leq Cq_{\mathsf{hash}} - 1$, we relax the problem to minimizing

$$\Psi = \sum_{i=0}^{Cq_{\mathsf{hash}}-1} P_i[(t_i - 1)(1 - e^{-q_{\mathsf{hash}}d}) + (1 - e^{-[i-q_{\mathsf{hash}}(t_i-1)]d})]$$

under the same constraints that $\sum_{i=0}^{Cq_{\mathsf{hash}}-1} P_i i = \beta$, $\sum_{i=0}^{Cq_{\mathsf{hash}}-1} P_i = 1$, and $P_i \geq 0$.

The optimal value of the relaxed problem will be no more than the optimal value of the original problem. To solve the relaxed problem, let

$$\Psi_s = \sum_{i=(s-1)q_{\mathsf{hash}}}^{sq_{\mathsf{hash}}-1} P_i[(s - 1)(1 - e^{-q_{\mathsf{hash}}d}) + (1 - e^{-[i-q_{\mathsf{hash}}(s-1)]d})]$$

for $1 \leq s \leq C$. Clearly, $\Psi = \sum_{s=1}^{C} \Psi_s$.

Suppose a set of $\{\gamma_s\}_s$ and $\{\beta_s\}_s$ are given where $\sum_{s=1}^{C} \gamma_s = 1$ and $\sum_{s=1}^{C} \beta_s = \beta$. The set of $\{\gamma_s\}_s$ and $\{\beta_s\}_s$ are called *feasible* if it is possible to find $\{P_i\}_i$ such that

$$\sum_{i=(s-1)q_{\mathsf{hash}}}^{sq_{\mathsf{hash}}-1} P_i = \gamma_s,$$

and

$$\sum_{i=(s-1)q_{\mathsf{hash}}}^{sq_{\mathsf{hash}}-1} P_i i = \beta_s$$

for all $1 \leq s \leq C$. Note that $\{\gamma_s\}_s$ and $\{\beta_s\}_s$ are feasible if and only if

$$(s - 1)q_{\mathsf{hash}}\gamma_s \leq \beta_s \leq (sq_{\mathsf{hash}} - 1)\gamma_s.$$

When $\{\gamma_s\}_s$ and $\{\beta_s\}_s$ are given and are feasible, to minimize Ψ is to minimize each individual Ψ_s. Note that

$$\begin{aligned}\Psi_s &= \gamma_s[(s - 1)(1 - e^{-q_{\mathsf{hash}}d}) + 1] - \sum_{i=(s-1)q_{\mathsf{hash}}}^{sq_{\mathsf{hash}}-1} P_i e^{-[i-q_{\mathsf{hash}}(s-1)]d} \\ &= \gamma_s[(s - 1)(1 - e^{-q_{\mathsf{hash}}d}) + 1] - \sum_{h=0}^{q_{\mathsf{hash}}-1} P_{q_{\mathsf{hash}}(s-1)+h} e^{-hd},\end{aligned}$$

where $h = i - q_{\mathsf{hash}}(s - 1)$. Note that if

$$\sum_{i=(s-1)q_{\mathsf{hash}}}^{sq_{\mathsf{hash}}-1} P_i i = \beta_s,$$

then

$$\sum_{h=0}^{q_{\mathsf{hash}}-1} P_{q_{\mathsf{hash}}(s-1)+h} h = \beta_s - (s - 1)q_{\mathsf{hash}}\gamma_s.$$

Denote the minimum value of Ψ_s for given γ_s and β_s as $\Psi_s^{\gamma_s,\beta_s}$. Applying Lemma **??**,

$$\begin{aligned}\Psi_s^{\gamma_s,\beta_s} &= \gamma_s[(s-1)(1-e^{-q_{\mathsf{hash}}d})+1]-\gamma_s+[\frac{1-e^{-(q_{\mathsf{hash}}-1)d}}{q_{\mathsf{hash}}-1}][\beta_s-(s-1)q_{\mathsf{hash}}\gamma_s] \\ &= \gamma_s(s-1)[\frac{q_{\mathsf{hash}}e^{-(q_{\mathsf{hash}}-1)d}}{q_{\mathsf{hash}}-1}-e^{-q_{\mathsf{hash}}d}-\frac{1}{q_{\mathsf{hash}}-1}]+[\frac{1-e^{-(q_{\mathsf{hash}}-1)d}}{q_{\mathsf{hash}}-1}]\beta_s\end{aligned}$$

Let

$$a=\frac{q_{\mathsf{hash}}e^{-(q_{\mathsf{hash}}-1)d}}{q_{\mathsf{hash}}-1}-e^{-q_{\mathsf{hash}}d}-\frac{1}{q_{\mathsf{hash}}-1}$$

and

$$b=\frac{1-e^{-(q_{\mathsf{hash}}-1)d}}{q_{\mathsf{hash}}-1},$$

we have

$$\Psi\geq\sum_{s=1}^{C}\Psi_s^{\gamma_s,\beta_s}=b\beta+a\sum_{s=1}^{C}(s-1)\gamma_s.$$

We also note that $a<0$, which is because function

$$f(x)=\frac{q_{\mathsf{hash}}e^{-(q_{\mathsf{hash}}-1)x}}{q_{\mathsf{hash}}-1}-e^{-q_{\mathsf{hash}}x}-\frac{1}{q_{\mathsf{hash}}-1}$$

is 0 when $x=0$, while $f'(x)<0$ for $x>0$. Therefore, finding the minimum value of Ψ is equivalent to finding a set of feasible $\{\gamma_s\}_s$ and $\{\beta_s\}_s$ such that $\sum_{s=1}^{C}(s-1)\gamma_s$ is maximized.

We consider the problem of maximizing

$$R=\sum_{s=1}^{C}(s-1)\gamma_s$$

subject to the constraints that

$$\begin{aligned}&(s-1)q_{\mathsf{hash}}\gamma_s\leq\beta_s\leq(sq_{\mathsf{hash}}-1)\gamma_s,\\ &\sum_{s=1}^{C}\gamma_s=\gamma,\\ &\sum_{s=1}^{C}\beta_s=\beta,\end{aligned}$$

where $0\leq\gamma\leq1$ and $0\leq\beta\leq\gamma(Cq_{\mathsf{hash}}-1)$. Denote the maximum value of R as R^*. We claim that

- If $(C-1)q_{\mathsf{hash}}\gamma<\beta$, $R^*=\gamma(C-1)$ and the optimal is achieved when $\gamma_C=\gamma$, $\beta_C=\beta$, while $\gamma_s=0$ and $\beta_s=0$ for $1\leq s<C$;
- If $(C-1)q_{\mathsf{hash}}\gamma\geq\beta$, $R^*=\frac{\beta}{q_{\mathsf{hash}}}$, and the optimal value is achieved when $\gamma_1=\gamma-\frac{\beta}{(C-1)q_{\mathsf{hash}}}$, $\beta_1=0$, $\gamma_C=\frac{\beta}{(C-1)q_{\mathsf{hash}}}$, $\beta_C=\beta$, while $\gamma_s=0$ and $\beta_s=0$ for $1<s<C$.

To show this, we use induction on C. First, when $C=2$, $W=\gamma_2$. Note that

- If $q_{\mathsf{hash}}\gamma < \beta$, we can let $\gamma_2 = \gamma$, $\beta_2 = \beta$, while $\gamma_1 = 0$ and $\beta_1 = 0$, in which case γ_2 is maximized, while all constraints are satisfied;
- If $q_{\mathsf{hash}}\gamma \geq \beta$, note that for any given β_2, $\gamma_2 \leq \frac{\beta_2}{q_{\mathsf{hash}}} \leq \frac{\beta}{q_{\mathsf{hash}}}$. When $\beta \leq q_{\mathsf{hash}}\gamma$, we may let $\gamma_1 = \gamma - \frac{\beta}{q_{\mathsf{hash}}}$, $\beta_1 = 0$, $\gamma_2 = \frac{\beta}{q_{\mathsf{hash}}}$, $\beta_2 = \beta$, such that all constraints are satisfied, while γ_2 is maximized.

Therefore, our claim is true when $C = 2$. Suppose the claim is true till $C = j$. When $C = j + 1$,

- If $jq_{\mathsf{hash}}\gamma < \beta$, we may let $\gamma_{j+1} = \gamma$, $\beta_{j+1} = \beta$, and let $\gamma_s = 0$ and $\beta_s = 0$ for $1 \leq s \leq j$, such that all constraints are satisfied. In this case, $R = j\gamma$. Since $R \leq j\gamma$, we have $R^* = j\gamma$.
- If $jq_{\mathsf{hash}}\gamma \geq \beta$, given any valid assignment $\{\gamma_s\}_s$ and $\{\beta_s\}_s$, let $\sum_{s=1}^{j}\gamma_s = \gamma'$ and $\sum_{s=1}^{j}\beta_s = \beta'$. γ' and β' must satisfy

  $$(\gamma - \gamma')jq_{\mathsf{hash}} \leq (\beta - \beta') \leq (\gamma - \gamma')[(j+1)q_{\mathsf{hash}} - 1]$$

  and

  $$\beta' \leq \gamma'(jq_{\mathsf{hash}} - 1).$$

  - If $(j-1)q_{\mathsf{hash}}\gamma' < \beta'$, based on the induction hypothesis, the maximum value of $\sum_{s=1}^{j}(s-1)\gamma_s$ is $(j-1)\gamma'$, and hence

    $$R \leq j\gamma - \gamma'.$$

    Because

    $$(\gamma - \gamma')jq_{\mathsf{hash}} \leq (\beta - \beta'),$$

    we have

    $$\gamma' \geq \gamma - \frac{\beta}{jq_{\mathsf{hash}}} + \frac{\beta'}{jq_{\mathsf{hash}}}.$$

    As $(j-1)q_{\mathsf{hash}}\gamma' < \beta'$, we have

    $$\gamma' > \gamma j - \frac{\beta}{q_{\mathsf{hash}}}.$$

    Therefore,

    $$R \leq \frac{\beta}{q_{\mathsf{hash}}}.$$

  - If $(j-1)q_{\mathsf{hash}}\gamma' \geq \beta'$, based on the induction hypothesis, the maximum value of $\sum_{s=1}^{j}(s-1)\gamma_s$ is $\frac{\beta'}{q_{\mathsf{hash}}}$, and hence

    $$R \leq \frac{\beta'}{q_{\mathsf{hash}}} + (\gamma - \gamma')j.$$

Since $(\gamma - \gamma')j \leq \frac{\beta - \beta'}{q_{\mathsf{hash}}}$, we have

$$R \leq \frac{\beta}{q_{\mathsf{hash}}}.$$

Note that R achieves $\frac{\beta}{q_{\mathsf{hash}}}$ when $\gamma_1 = \gamma - \frac{\beta}{jq_{\mathsf{hash}}}$, $\beta_1 = 0$, $\gamma_{j+1} = \frac{\beta}{jq_{\mathsf{hash}}}$, $\beta_{j+1} = \beta$, while $\gamma_s = 0$ and $\beta_s = 0$ for $1 < s \leq j$. Therefore, $R^* = \frac{\beta}{q_{\mathsf{hash}}}$.

Note that actually, in the first case when $jq_{\mathsf{hash}}\gamma < \beta$, $j\gamma \leq \frac{\beta}{q_{\mathsf{hash}}}$, therefore we also have $R^* \leq \frac{\beta}{q_{\mathsf{hash}}}$. Hence,

$$\Psi \geq b\beta + a\frac{\beta}{q_{\mathsf{hash}}} = \frac{\beta(1 - e^{-q_{\mathsf{hash}}d})}{q_{\mathsf{hash}}},$$

which completes our proof. ■

Similar to single puzzle case, we may now put things together. Suppose $\mathcal{G}$ has an advantage of σ in solving the puzzles when receiving $\omega(\mathcal{G})$ bits on average. Based on Theorem **??**, $\mathcal{B}_{\mathcal{G}}$ has an advantage of no less than $\sigma - \frac{Aq_{\mathsf{hash}}}{2^V}$ while receiving no more than $\omega(\mathcal{G}) + \frac{PLkAq_{\mathsf{hash}}}{2^V} + (V-1)P$ bits on average. Based on Theorem **??**, to achieve an advantage of at least $\sigma - \frac{Aq_{\mathsf{hash}}}{2^V}$, the optimal algorithm $\mathcal{B}^*$ must make at least $\frac{P(\sigma - \frac{Aq_{\mathsf{hash}}+1}{2^V})(L+1)}{2}$ informed hash queries. Based on Lemma **??**, also considering that $\mathcal{B}^*$ needs to receive only $k - V + 1$ bits per sequence, $\mathcal{B}^*$ receives at least $U(\frac{P(\sigma - \frac{Aq_{\mathsf{hash}}+1}{2^V})(L+1)}{2}) - PL(V-1)$ bits on average. Therefore,

Theorem 3.11: Suppose A adversaries are challenged with P puzzles. Suppose $\mathcal{G}$ solves the puzzle with probability no less than σ. We have

$$\omega(\mathcal{G}) \geq \frac{(1-\Upsilon)(1-\delta)nP(\sigma - \frac{Aq_{\mathsf{hash}}+1}{2^V})(L+1)(1-e^{-q_{\mathsf{hash}}k/n})}{2q_{\mathsf{hash}}} - P(L+1)(V-1) - \frac{PLkAq_{\mathsf{hash}}}{2^V}P$$

as $n \to \infty$ and $k \to \infty$, where q_{hash}, Υ, V, and δ are constants determined by the puzzle parameters.

IV. Discussions

In this section we discuss the bound and its practical implications. We begin by considering a simple strategy the adversaries may adopt to be compared with the bound.

A. A Simple Adversary Strategy

We note that there exists a simple strategy the adversaries may adopt, referred to as the *All-Or-Nothing (AON) Strategy*:

- When challenged with P puzzles, the adversaries flip a coin and decide whether or not to solve *all* puzzles. They attempt with probability ω; otherwise they simply ignore all puzzles. ω is called the *attempt probability*. If they decide to solve the puzzles, the adversities select $\frac{\tau P(L+1)}{2q_{\mathsf{hash}}}$ members, called the *active adversaries*, where τ is a constant slightly larger than 1, such as 1.01. Each of active adversary gets the entire content, and makes q_{hash} hash queries. For each puzzle, the active adversaries make hash queries for the sequences one by one until a $\mathsf{confirm}$ is obtained.

We note that the number of adversaries must be no less than $\frac{PL}{q_{\mathsf{hash}}}$, such that the adversaries can always find the required number of active adversaries. This is because the puzzle parameters should be selected such that $q_{\mathsf{hash}} \geq zL$ to ensure an honest prover can solve the puzzle.

We now analyze the performance of AON strategy. We argue that

Lemma 4.1: According AON strategy, as $P \rightarrow \infty$, the probability that the adversaries can solve the puzzles asymptotically approaches 1 if they decide to make the attempt.

Proof: Note that to obtain a $\mathsf{confirm}$ for a puzzle according to the AON strategy, the number of hash queries follows a uniform distribution in $[1, L]$ and is independent of other puzzles. Denote the total number of hash queries to obtain $\mathsf{confirm}$ for all puzzles as R. R is a random variable with mean $\frac{P(L+1)}{2}$ and variance $\frac{P(L^2-1)}{12}$. As $P \rightarrow \infty$, due to the Central Limit Theorem, the distribution of R approaches a Gaussian distribution. On the other hand, the active adversaries can make a total of $\frac{\tau P(L+1)}{2}$ hash queries. As $\tau > 1$, it can be verified that the adversaries can obtain $\mathsf{confirm}$ for all puzzles with probability asymptotically approaching 1. ■

Note that if the adversaries decide to make the attempt, they need to download $\frac{\tau n P(L+1)}{2q_{\mathsf{hash}}}$ bits in total. If they attempt to solve the puzzles for only ω fraction of the time, they download $\frac{\omega\tau n P(L+1)}{2q_{\mathsf{hash}}}$ bits in expectation.

The following theorem summaries the above discussions.

Theorem 4.2: According the AON strategy, as $P \rightarrow \infty$, the probability that the adversaries can solve all puzzles asymptotically approaches ω while downloading $\frac{\omega\tau n P(L+1)}{2q_{\mathsf{hash}}}$ bits in expectation.

Remark 4.3: ω is basically the advantage of the adversaries. Replacing σ in Theorem **??** with ω, noting that τ is only slightly greater than 1, the expected number of downloaded bits according to the AON strategy is a small fraction from the bound in Theorem **??** as $P \rightarrow \infty$, $n \rightarrow \infty$, and $k \rightarrow \infty$, if the following conditions are true:

1) Υ is small comparing to 1,
2) δ is small comparing to 1,
3) $e^{-q_{\mathsf{hash}}k/n}$ is small comparing to 1,
4) 2^V is much larger than Aq_{hash},
5) 2^V is no less than kAq_{hash},
6) V is much smaller than k.

B. Puzzle Parameter Space

We show that there are a wide range of puzzle parameters satisfying the conditions in Remark **??**. We consider $n \geq 10^7$ and $k \geq 10^4$, which can be considered as large enough for the conditions in the proofs to hold. In the following we use an example to illustrate the choices of parameters when $A \leq 10^6$; the parameters can be similarly determined for other values of A. Concerning the conditions,

- *For Conditions 1 and 2, we let* $\delta = 0.1$. The total number of sequences is AzL. We find numerically that when $A \leq 10^6$, $zL \leq 10^7$, $\delta = 0.1$, $n = 10^7$, $k = 10^4$, the probability that a combination of sequences do not satisfy the *Uniqueness Constraint* is no more than 10^{-10}, therefore Υ can be set to as small as 10^{-10}.
- *For Condition 3, we let* $q_{\mathsf{hash}} \geq \frac{4n}{k}$, noting that $e^{-4} = 0.018$.
- *For Conditions 4, 5, 6, we let* $V = 60$. Condition 4 is satisfied when $q_{\mathsf{hash}} \leq 10^7$. Condition 5 is satisfied when $k \leq 10^5$. Condition 6 is satisfied because $k \geq 10^4$.

Therefore, as long as

- $n \geq 10^7$, $10^5 \geq k \geq 10^4$, $10^7 \geq Lz$, $10^7 \geq q_{\mathsf{hash}} \geq \frac{4n}{k}$,

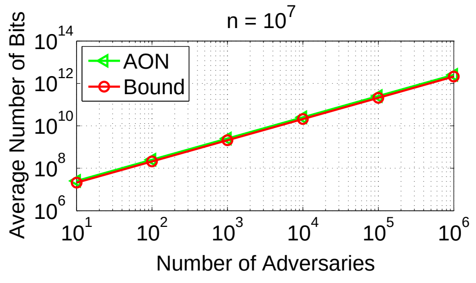
our bound is tight in the sense that it is a small fraction from the number of downloaded bits according to the AON strategy.

For example, Figure **??** shows the average number of downloaded bits by the AON strategy and the lower bound as a function of the number of adversaries for different content sizes, when $\sigma = 1$, $\tau = 1.01$, $\Upsilon = 10^{-10}$, $\delta = 0.1$, $V = 60$, $k = 10^4$, $q_{\mathsf{hash}} = 4n/k$ and $Lz = q_{\mathsf{hash}}/2$. We can see that they differ only by a small constant factor. We have tested other parameters satisfying the constraints and the results show similar trends.

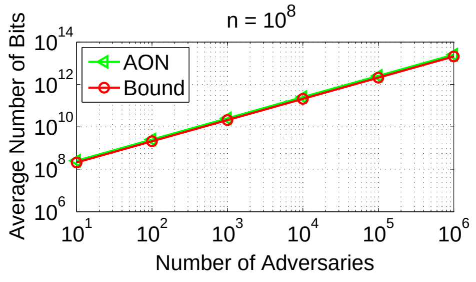
C. Puzzle Parameter Space in Practice

We also note that the parameter space is not restrictive in practice:

- n can be set to 10^7 or higher in practice. Considering the data rate of video broadcast, this leads to a challenge rate in the order of 10 seconds or higher, which we believe to be acceptable.
- k may be set to 10^4. k should be as small as possible, because a larger value of k results in a heavier load of the verifier.
- Concerning requirement $10^7 \geq Lz$, we note that for practical puzzles, Lz is unlikely to be greater than 10^7, due to the computation cost of hash queries. There are two time consuming tasks when making hash queries, namely the hash function call and the generation of the random indices. The



(a)



(b)

Fig. 1. Comparison of the average number of downloaded bits by the AON strategy and the bound. (a). $n = 10^7$. (b). $n = 10^8$.

choices of hash function and random number generator have been discussed in [**?**]. Basically, secure hash functions such as SHA-1 can be used as the hash function and block ciphers such as AES can be used to generate the random indices. The optimization of the puzzle implementation is out of the scope of this paper. We here show the speed of several machines in Emulab [**?**] when executing the SHA-1 hash and the AES encryption in the Openssl library [**?**], summarized in Table **??**, when the input to SHA-1 is 10^4 bits and the AES is 128 bits. We can see that it will take too long even for the fastest machine to finish 10^7 SHA-1 hash queries and the required AES calls for random numbers.

- Concerning requirement $10^7 \geq q_{\mathsf{hash}} \geq \frac{4n}{k}$, we note that $10^7 \geq q_{\mathsf{hash}}$ for the same reason as $10^7 \geq Lz$. As to $q_{\mathsf{hash}} \geq \frac{4n}{k}$, we note that if $n = 10^7$ and $k = 10^4$, this is equivalent to $q_{\mathsf{hash}} \geq 4 \times 10^3$. Based on our experiments in Emulab, it is possible for mainstream machines such as pc3000 and pc2000 to finish such number of queries in the order of seconds if the random number generation can be further optimized.

We also make some additional remarks concerning the requirements for puzzle parameters, which are not related to the validity of the bound but important in practice. First, the puzzle solving time should be in the order of seconds to accommodate issues such as network delay fluctuations. As a result, Lz

machine	CPU	SHA1	AES
pc3000	3.0GHz 64-bit	202165	4059157
pc2000	2.0GHz	71016	2605490
pc850	850MHz	39151	1086667
pc600	600MHz	29064	789624

TABLE 3

SHA-1 AND AES FUNCTION CALLS EXECUTED IN ONE SECOND.

should be selected such that it takes non-trivial time to make Lz hash queries. Second, as mentioned earlier, θ should be selected such that q_{hash} is greater than Lz.

V. RELATED WORK

Using puzzles has been proposed (e.g., in [**?**], [**?**], [**?**], [**?**], [**?**]) 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 if the claimed content transactions took place, where the ability to solve the puzzles is tied to the amount of content actually downloaded. As such, the construction of the bandwidth puzzle is different from existing puzzles.

Proofs of Data Possession (PDP) (e.g., [**?**], [**?**], [**?**], [**?**]) and Proofs of Retrievability (POR) (e.g., [**?**], [**?**], [**?**]) have been proposed, which allow a client to verify if the data has been modified in a remote store. As discussed in [**?**], the key differences between PDP/POR schemes and the bandwidth puzzle include the following. Fundamentally, the PDP/POR schemes and the bandwidth puzzle are designed for different systems. The PDP/POR schemes consider a user outsourcing the content to a server, and require the server to prove the possession or retrievability of the content while the user does not keep the content after outsourcing. The bandwidth puzzle considers a verifier and many provers where the provers may be colluding. The bandwidth puzzle assumes that the verifier keeps the content, and requires the provers to prove that they all possess the same content as an evidence for the claimed data transfer over the network. The PDP/POR schemes and the bandwidth puzzle use different techniques and are optimized for different system requirements. In particular, the PDP/POR schemes and the bandwidth puzzle have very different per challenge cost ratios between the verifier and the prover, where the costs include the challenge generation/verification cost for the verifier and the challenge response cost for the prover. In the existing PDP/POR schemes, the ratio is a constant not far from 1, i.e., the verifier and the prover incur similar costs per challenge, because the existing PDP/POR schemes are designed for the one-verifier-one-prover scenarios. In the bandwidth puzzle, because the verifier handles many provers simultaneously, the ratio is much smaller, i.e., 1:$L/2$ in expectation. The proof techniques for the existing PDP/POR schemes are also different from the proof techniques used in this paper, because the PDP/POR schemes considers a single prover while the bandwidth puzzle considers many provers who may be colluding.

In our earlier work [**?**], an upper bound was given on the expected number of puzzles that can be solved if the adversaries are allowed a certain number of hash queries and content queries. In this work, we remove the 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 [**?**], and new techniques have been 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 puzzles, there exists a lower bound on the average number of bits they have to download, regardless of the algorithm they adopt.

We published a shorter, conference version of this work in [**?**]. This paper contains substantially extended content compared to the conference version, especially the detailed analysis on the multiple adversary case.

VI. 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 with practical puzzle parameters. 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 choices of better puzzle parameters for practical systems.

References

[1] Emulab. http://www.emulab.net.

[2] Openssl. http://www.openssl.org/.

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

[4] 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.

[5] G. Ateniese, S. Kamara, and J. Katz. Proofs of Storage from Homomorphic Identification Protocols. Proc. of ASIACRYPT, 2009.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

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

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

[20] J. M. Wozencraft and I. M. Jacobs. *Principles of Communication Engineering*. Wiley, 1965.

[21] Z. Zhang. A new bound on the performance of the bandwidth puzzle. In *IEEE Globecom*, 2011.

Zhenghao Zhang (M'02) received his B.Eng. and M.S. degrees in electrical engineering from Zhejiang University, Hangzhou, China, in 1996 and 1999, respectively. He received his Ph.D. degree in electrical engineering from the State University of New York at Stony Brook in 2006.

From 1999 to 2001, he worked in industry as an embedded system Software Engineer. From 2006 to 2007, he was a Postdoctoral Researcher in the Computer Science Department at Carnegie Mellon University. Currently, he is an Assistant Professor in the Computer Science Department at Florida State University, Tallahassee, FL. His research interests include wireless networks, network security, and high speed optical networks.