

MIT Open Access Articles

*How Asymmetry Helps Buffer Management:
Achieving Optimal Tail Size in Cup Games*

The MIT Faculty has made this article openly available. **Please share** how this access benefits you. Your story matters.

Citation: Kuszmaul, William. 2021. "How Asymmetry Helps Buffer Management: Achieving Optimal Tail Size in Cup Games."

As Published: <https://doi.org/10.1145/3406325.3451033>

Publisher: ACM|Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing

Persistent URL: <https://hdl.handle.net/1721.1/146003>

Version: Final published version: final published article, as it appeared in a journal, conference proceedings, or other formally published context

Terms of use: Creative Commons Attribution 4.0 International license



How Asymmetry Helps Buffer Management: Achieving Optimal Tail Size in Cup Games

William Kuszmaul*

kuszmaul@mit.edu

Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

ABSTRACT

The cup game on n cups is a multi-step game with two players, a filler and an emptier. At each step, the filler distributes 1 unit of water among the cups, and then the emptier selects a single cup to remove (up to) 1 unit of water from.

There are several objective functions that the emptier might wish to minimize. One of the strongest guarantees would be to minimize *tail size*, which is defined to be the number of cups with fill 2 or greater. A simple lower-bound construction shows that the optimal tail size for deterministic emptying algorithms is $\Theta(n)$, however.

We present a simple randomized emptying algorithm that achieves tail size $\tilde{O}(\log n)$ with high probability in n for poly n steps. Moreover, we show that this is tight up to doubly logarithmic factors. We also extend our results to the *multi-processor cup game*, achieving tail size $\tilde{O}(\log n + p)$ on p processors with high probability in n . We show that the dependence on p is near optimal for any emptying algorithm that achieves polynomial-bounded backlog.

A natural question is whether our results can be extended to give *unending guarantees*, which apply to arbitrarily long games. We give a lower bound construction showing that no monotone memoryless emptying algorithm can achieve an unending guarantee on either tail size or the related objective function of backlog. On the other hand, we show that even a very small (i.e., $1/\text{poly } n$) amount of resource augmentation is sufficient to overcome this barrier.

CCS CONCEPTS

• Theory of computation → Online algorithms.

KEYWORDS

cup games, tail size, load balancing, randomized algorithms

*Supported by an NSF GRFP fellowship and a Fannie and John Hertz Fellowship. Research was partially sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.



This work is licensed under a Creative Commons Attribution International 4.0 License.

STOC '21, June 21–25, 2021, Virtual, Italy

© 2021 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-8053-9/21/06.

<https://doi.org/10.1145/3406325.3451033>

ACM Reference Format:

William Kuszmaul. 2021. How Asymmetry Helps Buffer Management: Achieving Optimal Tail Size in Cup Games. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21)*, June 21–25, 2021, Virtual, Italy. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3406325.3451033>

1 INTRODUCTION

At the start of the *cup game on n cups*, there are n empty cups. In each step of the game, a *filler* distributes 1 unit of water among the cups, and then an *emptier* removes (up to) 1 unit of water from a single cup of its choice. The emptier aims to minimize some measure of “behind-ness” for cups in the system (e.g., the height of the fullest cup, or the number of cups above a certain height). If the emptier’s algorithm is randomized, then the filler is an *oblivious adversary*, meaning it cannot adapt to the behavior of the emptier.

The cup game naturally arises in the study of processor scheduling, modeling a problem in which n tasks each receive work at varying rates, and a scheduler must pick one task to schedule at each time step [1, 6–8, 18, 22, 25, 28–33]. The cup game has also found numerous applications to network-switch buffer management [4, 21, 23, 35], quality of service guarantees [1, 7, 29], and data structure deamortization [2, 3, 10, 17–19, 24, 26, 34].

Bounds on Backlog. Much of the work on cup games has focused on bounding the *backlog* of the system, which is defined to be the amount of water in the fullest cup.

Research on bounding backlog has spanned five decades [1, 6–8, 11, 17, 18, 22, 25, 27–33]. Much of the early work focused on the *fixed-rate* version of the game, in which the filler places a fixed amount of water f_j into each cup j on every step [6–8, 22, 25, 28, 30–33]; in this case constant backlog is achievable [7, 31]. For the full version of the game, without fixed rates, constant backlog is not possible. In this case, the optimal deterministic emptying algorithm is known to be the greedy emptying algorithm, which always empties from the fullest cup, and which achieves backlog $O(\log n)$ [1, 17]. If the emptier is permitted to use a randomized algorithm, then it can do much better, achieving an asymptotically optimal backlog of $O(\log \log n)$ for poly n steps with high probability [11, 18, 27].

A Strong Guarantee: Small Tail Size. The *tail size* of a cup game at each step is the number of cups containing at least some constant c amount of water. For the guarantees in this paper, c will be taken to be 2.

A guarantee of small tail size is particularly appealing for scheduling applications, where cups represent tasks and water represents work that arrives to the tasks over time. Whereas a bound of b on

backlog guarantees that the furthest behind worker is only behind by at most b , it says nothing about the *number* of workers that are behind by b . In contrast, a small bound on tail size ensures that *almost no workers* are behind by more than $O(1)$.

The main result in this paper is a randomized emptying algorithm that achieves tail size $O(\log n \log \log n)$. The algorithm also simultaneously optimizes backlog, keeping the maximum height at $O(\log \log n)$. As a result the *total amount of water* above height 2 in the system is $\tilde{O}(\log n)$ with high probability. In contrast, the best possible deterministic emptying algorithm allows for up to $n^{1-\varepsilon}$ cups to *all* have fills $\Omega(\log n)$ at once (see the lower-bound constructions discussed in [18] and [12]).

The problems of optimizing tail size and backlog are closely related to the problem of optimizing the *c-shifted ℓ_p norm* of the cup game. Formally, the *c-shifted ℓ_p norm* is given by

$$\left(\sum_{i=1}^n \max(f_i - c, 0)^p \right)^{1/p},$$

where f_i is the fill of cup i .¹ The problem of bounding backlog corresponds to the problem of optimizing the ℓ_∞ norm of the cup game, and the problem of bounding tail size corresponds to bounding the 2-shifted ℓ_1 norm. By optimizing both metrics simultaneously our algorithm has the desirable property that it also achieves a bound of $\tilde{O}(\log n)$ for the 2-shifted ℓ_p norm for *any* $p \in O(1)$, which is optimal up to doubly logarithmic factors.

Past Work on Tail Size Using Beyond-Worst-Case Analysis.

To approach the combinatorial difficulty of analyzing cup games, past works have often turned to various forms of beyond worst-case analysis (e.g., smoothed analysis [11], semi-clairvoyance [29], resource augmentation [11, 17, 18, 29]). The most successful of these approaches has arguably been resource augmentation. In the cup game with ε resource augmentation, the filler is permitted to distribute at most $1-\varepsilon$ water into cups at each step (rather than 1 full unit), giving the emptier a small advantage. Resource augmentation can also be studied in a more extreme form by allowing the emptier to fully empty a cup on each step, rather than simply removing a single unit [17, 18]. Resource augmentation significantly simplifies the task of analyzing cup games — for example, there was nearly a 30 year gap between the first randomized bounds on backlog with resource augmentation [18] versus the first bounds without resource augmentation [27].

Currently, the best known guarantees with resource augmentation are achieved by the algorithm of [11] which, using $\varepsilon = 1/\text{polylog } n$, limits backlog to $O(\log \log n)$ and tail size to $\text{polylog } n$ (with high probability).

The algorithm, which is called the *smoothed greedy algorithm*, begins by randomly perturbing the starting state of the system, and then following a variant of the deterministic greedy emptying algorithm. Roughly speaking, the random perturbations at the beginning of the game allow for the number of cups X_t containing more than 1 unit of water after each step t to be modeled by a

¹When $p \neq \infty$, in order for the *c-shifted ℓ_p norm* to be interesting, it is necessary to require that $c \geq \Omega(1)$, since trivial filling strategies can achieve fill $\Omega(1)$ in $\Theta(n)$ cups deterministically.

biased random walk X_1, X_2, \dots , where $\Pr[X_i = X_{i-1} + 1] = 1/2 - \varepsilon$ and $\Pr[X_i = \max(X_{i-1} - 1, 0)] = 1/2 + \varepsilon$. This is where the resource augmentation plays a critical role, since it introduces a bias to the random walk which pushes the walk near 0 at all times. In contrast, without resource augmentation, the random walk is unbiased.

Subsequent work [27] showed that the algorithm's guarantees on backlog continue to hold without resource augmentation (at least, for a polynomial number of steps). More generally, the analysis bounds the number of cups at a given height α by roughly $n^{1/2^{\Omega(\alpha)}}$, which in turn implies an arbitrarily small polynomial tail size. Whether or not a subpolynomial tail size can be achieved without resource augmentation has remained open.

This Paper: The Asymmetric Smoothed Greedy Algorithm.

We show that resource augmentation is not needed to bound tail size. We present a randomized emptying algorithm (called the *asymmetric smoothed greedy algorithm*) that achieves tail size $\tilde{O}(\log n)$ with high probability in n after each of the first $\text{poly } n$ steps of a cup game. We prove that the algorithm is nearly optimal, in that any emptying algorithm must allow for a tail size of $\tilde{\Omega}(\log n)$ with probability at least $\frac{1}{\text{poly } n}$.

The analysis of the algorithm takes an indirect approach to bounding tail size. Rather than examining tail size directly, we instead prove that the use of randomness by the algorithm makes the state of the cups at each step “difficult” for the filler to accurately predict. We call this the *unpredictability guarantee*. We then show that any greedy-like algorithm that satisfies the unpredictability guarantee is guaranteed to have tail size $\tilde{O}(\log n)$ and backlog $O(\log \log n)$.

Algorithmically, the key to achieving the unpredictability guarantee is to add a small amount of asymmetry to the smoothed greedy emptying algorithm. When choosing between cups that have 2 or more units of water, the emptier follows the standard smoothed greedy algorithm; but when choosing between cups whose fills are between 1 and 2, the emptier ignores the specifics of how much water is in each cup, and instead chooses between the cups based on random priorities that are assigned to the cups at the beginning of the game.

Intuitively, the asymmetric treatment of cups ensures that there is a large collection (of size roughly $n/2$) of randomly selected cups that are “almost always” empty. The fact that the emptier doesn't know which cups these are then implies the unpredictability guarantee. Proving this intuition remains highly nontrivial, however, and requires several new combinatorial ideas.

Multi-Processor Guarantees. The cup game captures a scheduling problem in which a single processor must pick one of n tasks to make progress on in each time step. The multi-processor version of this scheduling problem is captured by the *p-processor cup game* [1, 7, 11, 27, 29, 31]. In each step of the *p-processor cup game*, the filler distributes p units of water among cups, and the emptier removes 1 unit of water from (up to) p different cups. Because the emptier can remove at most 1 unit of water from each cup at each step, an analogous constraint is also placed on the filler, requiring that it places at most 1 unit of water into each cup at each step.

A key feature of the p -processor cup game is that the emptier is required to remove water from p *distinct* cups in each step, even if the vast majority of water is contained in fewer than p cups.

Until recently, establishing any nontrivial bounds on backlog in the multi-processor cup game remained an open problem, even with the help of resource augmentation. Recent work by Bender et al. [11] (using resource augmentation) and then by Kuszmaul [27] (without resource augmentation) established bounds on backlog closely matching those for the single-processor game.

By extending our techniques to the multi-processor setting, we construct a randomized emptying algorithm achieves tail size $\tilde{O}(\log n + p)$ with high probability in n after each of the first poly n steps of a p -processor cup game. Moreover, we show that the dependence on p is near optimal for any backlog-bounded algorithm (i.e., any algorithm that achieves backlog poly n or smaller).

Lower Bounds Against Unending Guarantees. In the presence of resource augmentation $\varepsilon = 1/\text{polylog } n$, the smoothed greedy emptying algorithm is known to provide an *unending guarantee* [11], meaning that the high-probability bounds on backlog and tail size continue to hold even for *arbitrarily* large steps t .

A natural question is whether unending guarantees can also be achieved *without* the use of resource augmentation. It was previously shown that, when $p \geq 2$, the smoothed greedy algorithm does not offer unending guarantees [27]. Analyzing the single-processor game has remained an open question, however.

We give a lower bound construction showing that neither the smoothed greedy algorithm nor the asymmetric smoothed greedy algorithm offer unending guarantees for the single-processor cup game without the use of resource augmentation. Even though resource augmentation $\varepsilon > 0$ is needed for the algorithms to achieve unending guarantees, we show that the amount of resource augmentation required is very small. Namely, $\varepsilon = 1/2^{\text{polylog } n}$ is both sufficient and necessary for the asymmetric smoothed greedy algorithm to offer unending guarantees on both tail size and backlog.

We generalize our lower-bound construction to work against *any emptying algorithm* that is both monotone and memoryless, including emptying algorithms that are equipped with a clock. We show that no such emptying algorithm can offer an unending guarantee of $o(\log n)$ backlog in the single-processor cup game, and that any unending guarantee of polylog n tail size must come of the cost of polynomial backlog.

We call the filling strategy in our lower bound construction the **fuzzing algorithm**. The fuzzing algorithm takes a very simple approach: it randomly places water into a pool of cups, and shrinks that pool of cups very slowly over time. The fact that gradually shrinking random noise represents a worst-case workload for cup games suggests that real-world applications of cup games (e.g., processor scheduling, network-switch buffer management, etc.) may be at risk of experiencing “aging” over time, with the performance of the system degrading due to the impossibility of strong unending guarantees.

Related Work on Other Variants of Cup Games. Extensive work has also been performed on other variants of the cup game.

Bar-Noy et al. [5] studied the backlog for a variant of the single-processor cup game in which the filler can place arbitrarily large integer amounts of water into cups at each step. Rather than directly bounding the backlog, which would be impossible, they show that the greedy emptying algorithm achieves competitive ratio $O(\log n)$, and that this is optimal for both deterministic and randomized online emptying algorithms. Subsequent work has also considered weaker adversaries [16, 20].

Several papers have also explored variants of cup games in which cups are connected by edges in a graph, and in which the emptier is constrained by the structure of the graph [12–15]. This setting models multi-processor scheduling with conflicts between tasks [14, 15] and some problems in sensor radio networks [12].

Cup games have also been used to model memory-access heuristics in databases [9]. Here, the emptier is allowed to completely empty a cup at each step, but the water from that cup is then “recycled” among the cups according to some probability distribution. The emptier’s goal is achieve a large recycling rate, which is the average amount of water recycled in each step.

Closely related to the study of cup games is the problem of **load balancing**, in which one must assign balls to bins in order to minimize the number of balls in the fullest bin. In the classic load balancing problem, n balls arrive over time, and each ball comes with a selection of d random bins (out of n bins) in which it can potentially be placed. The load balancing algorithm gets to select which of the d bins to place the ball in, and can, for example, always select the bin with the fewest balls. But what should the algorithm do when choosing between bins that have the same number of balls? In this case, Vöcking famously showed that the algorithm should always break ties in the same direction [37], and that this actually results in an asymptotically better bound on load than if one breaks ties arbitrarily. Interestingly, one can think of the asymmetry used in Vöcking’s algorithm for load balancing as being analogous to the asymmetry used in our algorithm for the cup game: in both cases, the algorithm always breaks ties in the same random direction, although in our result, the way that one should define a “tie” is slightly nonobvious. In the case of Vöcking’s result, the asymmetry is known to be necessary in order to get an optimal algorithm [37]; it remains an open question whether the same is true for the problem of bounding tail size in cup games.

Related Work on the Roles of Backlog and Tail Size in Data Structures. Bounds on backlog have been used extensively in data-structure deamortization [2, 3, 17–19, 24, 26, 34], where the scheduling decisions by the emptier are used to decide how a data structure should distribute its work.

Until recently, the applications focused primarily on in-memory data structures, since external-memory data structures often cannot afford the cost of a buffer overflowing by an $\omega(1)$ factor. Recent work shows how to use bounds in tail-size in order to solve this problem, and presents a new technique for applying cup games to external-memory data structures [10]. A key insight is that if a cup game has small tail size, then the water in “overflowed cups” (i.e., cups with fill more than $O(1)$) can be stored in a small in-memory cache. The result is that every cup consumes exactly $\Theta(1)$ blocks in external memory, meaning that each cup can be read/modified

by the data structure in $O(1)$ I/Os. This insight was recently applied to external-memory dictionaries in order to eliminate flushing cascades in write optimized data structures [10].

Outline. The paper is structured as follows. Section 2 describes a new randomized algorithm that achieves small tail size without resource augmentation. Section 3 gives a technical overview of the algorithm’s analysis and of the other results in this paper. Section 4 then presents the full analysis of the algorithm and Section 5 presents (nearly) matching lower bounds. Finally, Section 6 gives lower bounds against unending guarantees and analyzes the amount of resource augmentation needed for such guarantees.

Conventions. Although in principle an arbitrary constraint height c can be used to determine which cups contribute to the tail size, all of the algorithms in this paper work with $c = 2$. Thus, throughout the rest of the paper, we define the **tail size** to be the number of cups with height 2 or greater.

As a convention, we say that an event occurs with **high probability in n** , if the event occurs with probability at least $1 - \frac{1}{n^c}$ for an arbitrarily large constant c of our choice. The constant c is allowed to affect other constants in the statement. For example, an algorithm that achieves tail size $c \log n$ with probability $\frac{1}{n^c}$ is said to achieve tail size $O(\log n)$ with high probability in n .

2 THE ASYMMETRIC SMOOTHED GREEDY ALGORITHM

Past work on randomized emptying algorithms has focused on analyzing the **smoothed greedy algorithm** [11, 27]. The algorithm begins by randomly perturbing the starting state of the system: the emptier places a random offset r_j of water into each cup j , where the r_j ’s are selected independently and uniformly from $[0, 1)$. The emptier then follows a greedy emptying algorithm, removing water from the fullest cup at each step. If the fullest cup contains fill less than 1, however, then the emptier skips its turn. This ensures that the **fractional** amount of water in each cup j (i.e., the amount of water modulo 1) is permanently randomized by the initial offset r_j . The randomization of the fractional amounts of water in each cup has been critical to past randomized analyses [11, 27], and continues to play an important (although perhaps less central) role in this paper.

This paper introduces a new variant of the smoothed greedy algorithm that we call the **asymmetric smoothed greedy algorithm**. The algorithm assigns a random priorities $p_j \in [0, 1)$ to each cup j (at the beginning of the game) and uses these to “break ties” when cups contain relatively small amounts of water. Interestingly, by always breaking these ties in the same direction, we change the dynamics of the game in a way that allows for new analysis techniques. We describe the algorithm in detail below.

Algorithm Description. At the beginning of the game, the emptier selects random offsets $r_j \in [0, 1)$ independently and uniformly at random for each cup j . Prior to the game beginning, r_j units of water are placed in each cup j . This water is for “bookkeeping” purposes only, and need not physically exist. During initialization, the

emptier also assigns a random **priority** $p_j \in [0, 1)$ independently and uniformly at random to each cup j .

After each step t , the emptier selects (up to) p different cups to remove 1 unit of water from as follows. If there are p or more cups containing 2 or more units of water, then the emptier selects the p fullest such cups. Otherwise, the emptier selects all of the cups containing 2 or more units of water, and then resorts to cups containing fill in $[1, 2)$, choosing between these cups based on their priorities p_j (i.e., choosing cups with larger priorities over those with smaller priorities). The emptier never removes water from any cup containing less than 1 unit of water.

Threshold Crossings and a Threshold Queue. When discussing the algorithm, several additional definitions and conventions are useful. We say that **threshold (j, i) is crossed** if cup j contains at least i units of water for positive integer i . When $i = 1$, the threshold (j, i) is called a **light threshold**, and otherwise (j, i) is called a **heavy threshold**. One interpretation of the emptying algorithm is that there is a queue Q of thresholds (j, i) that are currently crossed. Whenever the filler places water into cups, this may add thresholds (j, i) to the queue. And whenever the emptier removes water from some cup j , this removes some threshold (j, i) from the queue. When selecting thresholds to remove from the queue, the emptier prioritizes heavy thresholds over light ones. Within the heavy thresholds, the emptier prioritizes based on cup height, and within the light thresholds the emptier prioritizes based on cup priorities p_j .

As a convention, we say that a cup j is **queued** if $(j, 1)$ is in Q (or, equivalently, if (j, i) is in the queue for any i). The emptier is said to **dequeue** cup j whenever threshold $(j, 1)$ is removed from the queue. The **size** of the queue Q refers to the number of thresholds in the queue (rather than the number of cups).

3 TECHNICAL OVERVIEW

In this section we give an overview of the analysis techniques used in the paper. We begin by discussing the analysis of the asymmetric smoothed greedy algorithm. To start, we focus our analysis on the single-processor cup game, in which $p = 1$.

The Unpredictability Guarantee. At the heart of the analysis is what we call the **unpredictability guarantee**, which, roughly speaking, establishes that the filler cannot predict large sets of cups that will all be over-full at the same time as one-another. We show that if an algorithm satisfies a certain version of the unpredictability guarantee, along with certain natural “greedy-like” properties, then the algorithm is guaranteed to exhibit a small tail size.

Formally, we say that an emptying algorithm satisfies **R -unpredictability** at a step t if for any oblivious filling algorithm, and for any set of cups S whose size is a sufficiently large constant multiple of R , there is high probability in n that at least one cup in S has fill less than 1 after step t . In other words, for any polynomial $f(n)$, there exists a constant c such that: for each set $S \subseteq [n]$ of cR cups, the probability that every cup in S has height 1 or greater at step t is at most $1/f(n)$.

How R -Unpredictability Helps. Rather than proving that R -unpredictability causes the tail size to stay small, we instead show the contrapositive. Namely, we show that if there is a filling strategy that achieves a large tail size, the strategy can be adapted to instead violate R -unpredictability.

Suppose that the filler is able to achieve tail size cR at some step t , where c is a large constant. Then during each of the next cR steps, the emptier will remove water from cups containing fill 2 or more (here, we use the crucial fact that the emptier always prioritizes cups with fills 2 or greater over cups with fills smaller than 2). This means that, during steps $t + 1, \dots, t + cR$, the set of cups with fill 1 or greater is monotonically increasing. During these steps the filler can place 1 unit of water into each of the cups $1, 2, \dots, cR$ in order to ensure that these cups all contain fill 1 or greater after step $t + cR$. Thus the filler can transform the initial tail size of cR into a large set of cups $S = \{1, 2, \dots, cR\}$ that all have fill 1 or greater. In other words, any filling strategy for achieving large tail size (at some step t) can be harnessed to violate R -unpredictability (at some later step $t + cR$).

The directness of the argument above may seem to suggest that in order to prove the R -unpredictability, one must first (at least implicitly) prove a bound on tail size. A key insight in this paper is that the use of priorities in the asymmetric smoothed greedy algorithm allows for R -unpredictability to be analyzed as its own entity.

Our algorithm analysis establishes $\log n \log \log n$ unpredictability for the first poly n steps of any cup game, with high probability in n . This, in turn, implies a bound of $O(\log n \log \log n)$ on tail size.

Establishing Unpredictability. We prove that, out of the roughly $n/2$ cups j with priorities $p_j \geq 1/2$, at most $O(\log n \log \log n)$ of them are queued (i.e., contain fill 1 or greater) at a time, with high probability in n . Recall that the cups with priority $p_j \geq 1/2$ are prioritized by the asymmetric smoothed greedy algorithm when the algorithm is choosing between cups with fills in the interval $[1, 2)$. This preferential treatment does not extend the case where there are cups containing fill ≥ 2 , however. Remarkably, the limited preferential treatment exhibited by the algorithm is enough to ensure that the number of queued high-priority cups never exceeds $O(\log n \log \log n)$.

The bound of $O(\log n \log \log n)$ on the number of queued cups with priorities $\geq 1/2$ implies $\log n \log \log n$ -unpredictability as follows. For any fixed set S of cups, the number of cups j in S with priority $p_j \geq 1/2$ will be roughly $|S|/2$ with high probability in n . If $|S|/2$ is at least a sufficiently large constant multiple of $\log n \log \log n$, then the number of cups with $p_j \geq 1/2$ in S exceeds the *total* number of cups with $p_j \geq 1/2$ that are queued. Thus S must contain at least one non-queued cup, as required for the unpredictability guarantee.

In order to bound the number of queued cups with priority $p_j \geq 1/2$ by $O(\log n \log \log n)$, we partition the cups into $\Theta(\log \log n)$ **priority levels** based on their priorities p_j . Let q be a sufficiently large constant multiple of $\log \log n$. The priority level of a cup j is given by $\lfloor p_j \cdot q \rfloor + 1$. (Note that the priority levels are only needed in the analysis, and the algorithm does not have to know q .) We show that with high probability in n , there are never more than

$O(\log n \log \log n)$ queued cups with priority level $\geq q/2$. Note that, although we only care about bounding the number of queued whose priority-levels are in the top fifty percentile, our analysis will take advantage of the fact that the priorities p_j are defined at a high granularity (rather than, for example, being boolean).

The Stalled Emptier Problem. Bounding the number of queued cups with priority level greater than ℓ directly is difficult for the following reason: Over the course of a sequence of steps, the filler may cross many *light* thresholds cups with priority level greater than ℓ , while the emptier only removes *heavy* thresholds from Q (i.e., the emptier empties exclusively from cups of height 2 or greater). This means that, in a given sequence of steps, the number of queued cups with priority level greater than ℓ could increase substantially. We call this the **stalled emptier problem**. Note that the stalled emptier problem is precisely what enables the connection between tail size and R -unpredictability above, allowing the filler to transform large tail size into a violation of R -unpredictability. As a result, any analysis that directly considers the stalled emptier problem must also first bound tail-size, bringing us back to where we started.

Rather than bounding the number of queued cups with priority level greater than ℓ , we instead compare the number of queued cups at priority level greater than ℓ to the number at priority level ℓ . The idea is that, if the stalled-emptier problem allows for the number of queued priority-level greater than ℓ to grow large, then it will allow for the number of queued priority-level- ℓ cups to grow even larger. That is, without proving any absolute bound on the number of cups at a given priority level, we can still say something about the ratio of high-priority cups to low-priority cups in the queue.

To be precise, we prove that, whenever there are k queued cups at some priority level ℓ , there are at most $O(\sqrt{qk \log n} + \log n)$ queued cups at priority level $> \ell$ (recall that $q = \Theta(\log \log n)$ is the number of priority levels). Since the number of cups with priority level at least 1 is deterministically anchored at n , this allows for us to inductively bound the number of queued cups with large priority levels ℓ . In particular, the number of queued cups at priority level $q/2$ or greater never exceeds $O(\log n \log \log n)$.

Comparing the Number of Queued Cups with Priority Level ℓ Versus $> \ell$. Suppose that after some step t , there are some large number k of queued cups with priority level $\geq \ell$. We wish to show that almost all of these k cups have priority level exactly ℓ .

Before describing our approach in detail (which we do in the following two subheaders), we give an informal description of the approach. Let k_1 be number of priority-level- ℓ queued cups, and let k_2 be the number of priority-level-greater-than- ℓ queued cups. The only way that there can be a large number k_2 of priority-level-greater-than- ℓ cups queued is if they have all entered the queue since the last time that a level- ℓ cup was dequeued. This means that the size of Q has increased by at least k_2 since the last time that a priority-level- ℓ cup was dequeued. On the other hand, we show that priority-level- ℓ cups accumulate in Q at a much faster rate than the size of Q varies. In particular, we show that both the rate at which priority-level ℓ cups accumulate in Q and the rate at which Q 's size varies are controlled by what we call the “influence”

of a time-interval, and that the former is always much larger than the latter. This ensures that $k_1 \gg k_2$.

Note that the analysis avoids arguing directly the number of queued high-priority cups small, which could be difficult due to the stalled emptier problem. Intuitively, the analysis instead shows that low priority cups do a good job “pushing” the high priority cups out of the queue, ensuring that the ratio of low-priority cups (i.e., cups with priority level ℓ) to high-priority cups (i.e. cups with priority level $> \ell$) is always very large.

Relating the Number of High-Priority Queued Cups to

Changes in $|Q|$. Let t_0 be the most recent step $t_0 \leq t$ such that at least $k+1$ distinct cups C with priority level ℓ cross thresholds during steps t_0, \dots, t . (Recall that k is the number of queued cups with priority level $\geq \ell$ after step t .) One can think of the steps t_0, \dots, t as representing a long period of time in which many cups with priority level ℓ have the opportunity to accumulate in Q . We will now show that the use of priorities in the asymmetric smoothed greedy algorithm causes the following property to hold: The number of queued cups with priority level $> \ell$ after step t is bounded above by the amount that $|Q|$ varies during steps t_0, \dots, t .

Because Q contains only k queued cups with priority level $\geq \ell$ after step t , at least one cup from C must be dequeued during steps t_0, \dots, t (otherwise, Q would contain at least $|C| = k+1$ level- ℓ cups after step t). Let t^* be the final step in t_0, \dots, t out of those that dequeue a cup with priority level $\leq \ell$, and let Q_{t^*} and Q_t denote the queue after steps t^* and t , respectively.

By design, the only way that the asymmetric smoothed greedy algorithm can dequeue a cup with priority level $\leq \ell$ at step t^* is if the queue Q_{t^*} consists exclusively of light thresholds (i.e., thresholds of the form $(j, 1)$) for cups j with priority level $\leq \ell$. Moreover, the thresholds in Q_{t^*} must remain present in Q_t , since by the definition of t^* no cups with priority level $\leq \ell$ are dequeued during steps $t^* + 1, \dots, t$.

Since $Q_{t^*} \subseteq Q_t$ and Q_{t^*} contains only thresholds for cups with priority level $\leq \ell$, the total number of thresholds in Q_t for cups with priority level $> \ell$ is at most $|Q_t| - |Q_{t^*}|$. In other words, the only way that a large number of cups with priority level $> \ell$ can be queued after step t is if the size of Q varies by a large amount during steps t_0, \dots, t .

Although $t - t_0$ may be very large compared to k (e.g. $\text{poly } n$) we show that the amount by which $|Q|$ varies during steps t_0, \dots, t is guaranteed to be small as a function of k , bounded above by $O(\sqrt{kq \log n})$. This means that, out of the k cups with priority level $\geq \ell$ in Q_t , at most $O(\sqrt{kq \log n})$ of them can have priority level $\ell+1$ or larger.

The Influence Property: Bounding the Rate at which $|Q|$

Varies. The main tool in order to analyze the rate at which Q 's size varies is to analyze sequences of steps based on their *influence*. For sequence of steps I , the influence of I is defined to be $\sum_{j=1}^n \min(1, c_j(I))$, where $c_j(I)$ is the amount of water poured into each cup j during interval I . We show that, for any priority level ℓ and for any step interval I with influence $2rq$ for some r , either $r = O(\log n)$, or two important properties are guaranteed to hold with high probability:

- *Step interval I crosses thresholds in at least r cups with priority level ℓ .* This is true of any interval I with influence at least $2rq$ by a simple concentration-bound argument.
- *The size of Q varies by at most $O(\sqrt{qr \log n})$ during step interval I .* The key here is to show that, during each subinterval $I' \subseteq I$, the number of thresholds crossed by the filler is within $O(\sqrt{qr \log n})$ of $|I'|$. In order to do this, we take advantage of the initial random offsets r_j that are placed into each cup by the algorithm. If the filler puts some number $c_j(I')$ of units of water into a cup j during I' , then the cup j will deterministically cross $\lfloor c_j(I') \rfloor$ thresholds, and with probability $c_j(I') - \lfloor c_j(I') \rfloor$ will cross one additional threshold (with the outcome depending on the random value r_j). Since the influence of I' is at most $2rq$, we know that $\sum_j (c_j(I') - \lfloor c_j(I') \rfloor) \leq 2rq$. That is, if we consider only the threshold crossings that are not certain, then the number of them is a sum of independent 0-1 random variables with mean at most $2rq$. By a Chernoff bound, this number varies from its mean by at most $O(\sqrt{qr \log n})$, with high probability in n .

Combined, we call these the *influence property*. By a union bound, the influence property holds with high probability on all sub-sequences of steps during the cup game, and for all values r .

The influence property creates a link between the number of cups with priority level ℓ that cross thresholds during a sequence of steps I , and the amount by which $|Q|$ varies during steps I . Applying this link with $r = k+1$ to steps t_0, \dots, t , as defined above, implies that $|Q|$ varies by at most $O(\sqrt{kq \log n})$ during steps t_0, \dots, t . This, in turn, bounds the number of queued cups with priority level $\ell+1$ or larger by $O(\sqrt{kq \log n})$ after step t , completing the analysis.

Extending the Analysis to the Multi-Processor Cup Game.

The primary difficulty in analyzing the multi-processor cup game (i.e., when $p > 1$) is that the emptier must remove water from p different cups, even if almost all of the water in the system resides in fewer than p cups. For example, the emptier may dequeue a cup j even though there are up to $p-1$ other higher-priority cups that are still queued; furthermore, each of these higher-priority cups may contribute a large number of heavy thresholds to the queue Q .

We solve this issue by leveraging recent bounds on backlog for the p -processor cup game [27], which prove that the deterministic greedy emptying algorithm achieves backlog $O(\log n)$. This can be used to ensure that, for any $p-1$ cups that are queued, each of them can only contribute a relatively small number of thresholds to the queue Q . These “miss-behaving” thresholds can then be absorbed into the algorithm analysis.

Nearly Matching Lower Bounds on Tail Size. Our lower-bound constructions extend the techniques used in past works for backlog [11, 18, 27] in order to apply similar ideas to tail size. One of the surprising features of our lower bounds is that they continue to be nearly tight even in the multi-processor case — the same is not known to be true for backlog. We defer further discussion of the lower bounds to Section 5.

Lower Bounds Against Unending Guarantees. Finally, we consider the question of whether the analysis of the asymmetric smoothed greedy algorithm can be extended to offer an *unending guarantee*, i.e., a guarantee that for any step t , no matter how large, there a high probability at step t that the backlog and tail size are small.

We show that, without the use of resource augmentation, unending guarantees are not possible for the asymmetric smoothed greedy algorithm, or, more generally, for any monotone memoryless emptying algorithm. Lower bounds against unending guarantees have previously been shown for the multi-processor cup game [27], but remained open for the single-processor cup game.

The filling strategy, which we call the *fuzzing algorithm*, has a very simple structure: the filler spends a large number (i.e., $n^{\tilde{\Theta}(n)}$) of steps randomly placing water in multiples of $1/2$ into cups $1, 2, \dots, n$. The filler then disregards a random cup, which for convenience we will denote by n , and spends a large number of steps randomly placing water into the remaining cups $1, 2, \dots, n-1$. The filler then disregards another random cup, which we will call cup $n-1$, and spends a large number of steps randomly placing water into cups $1, 2, \dots, n-2$, and so on. We call the portion of the algorithm during which the filler is focusing on cups $1, 2, \dots, i$ the *i -cup phase*.

Rather than describe the analysis of the fuzzing algorithm (which is somewhat complicated), we instead give an intuition for why the algorithm works. For simplicity, suppose the emptier follows the (standard) smoothed greedy emptying algorithm.

Between the i -cup phase and the $(i-1)$ -cup phase, the filler disregards a random cup (that we subsequently call cup i). Intuitively, at the time that cup i is discarded, there is a roughly 50% chance that cup i has more fill than the average of cups $1, 2, \dots, i$. Then, during the $(i-1)$ -cup phase, there is a reasonably high probability that the filler at some point manages to make all of cups $1, 2, \dots, i-1$ have almost equal fills to one-another. At this point, the emptier will choose to empty out of cup i instead of cups $1, 2, \dots, i-1$. The fact that the emptier neglects cups $1, 2, \dots, i-1$ during the step, even though the filler places 1 unit of water into them, causes their average fill to increase by $1/(i-1)$. Since this happens with constant probability in every phase, the result is that, by the beginning of the \sqrt{n} -cup phase there are \sqrt{n} cups each with expected fill roughly

$$\Omega\left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{\sqrt{n}+1}\right) = \Omega(\log n).$$

Formalizing this argument leads to several interesting technical problems. Most notably, the cups $1, 2, \dots, i-1$ having *almost* equal fills (rather than exactly equal fills) may not be enough for cup i to receive the emptier's attention. Moreover, if we wish to analyze the *asymmetric* smoothed greedy algorithm or, more generally, the class of monotone memoryless algorithms, then cups are not necessarily judged by the emptying algorithm based on their fill heights, and may instead be selected based on an essentially arbitrary objective function that need not treat cups symmetrically. These issues are handled in Section 6 by replacing the notion of cups $1, 2, \dots, i-1$ having almost equal fills as each other with the notion of cups $1, \dots, i-1$ reaching a certain type of specially designed equilibrium state that interacts well with the emptier.

4 ALGORITHM ANALYSIS

In this section, we give the full analysis of the p -processor asymmetric smoothed greedy algorithm. The main result of the section is Theorem 4.10, which bounds the tail size of the game by $O(\log n \log \log n + p \log p)$ for the first poly n steps of the game with high probability in n .

In addition to using the conventions from Section 2 we find it useful to introduce one additional notation: for a sequence of steps I , define $c_j(I)$ to be the amount of water placed into cup j during I . We also continue to use the convention from Section 3 that q is a large constant multiple of $\log \log n$, and that each cup j is assigned a *priority level* given by $\lfloor p_j \cdot q \rfloor + 1$.

Recall that a cup j crosses a threshold (j, i) whenever the fill of cup j increases from some quantity $f < i$ to some quantity $f' \geq i$ for $i \in \mathbb{N}$. A key property of the smoothed greedy algorithm, which was originally noted by Bender et al. [11], is that the number of threshold crossings across any sequence of steps can be expressed using a sum of independent 0-1 random variables.² This remains true for the asymmetric smoothed greedy algorithm, and is formalized in Lemma 4.1.

Lemma 4.1 (Counting threshold crossings). *For a sequence of steps I , and for a cup j , the number of threshold crossings in cup j is $\lfloor c_j(I) \rfloor + X_j$, where X_j is a 0-1 random variable with mean $c_j(I) - \lfloor c_j(I) \rfloor$. Moreover, X_1, X_2, \dots, X_n are independent.*

PROOF. Recall that the emptier only removes water from a cup j if cup j contains at least 1 unit. Moreover, the emptier always removes exactly 1 unit of water from cups. Since threshold crossings in each cup j depend only on the fractional amount of water (i.e., the amount of water modulo 1) in the cup, the behavior of the emptier cannot affect when thresholds are crossed within each cup.

Let t_0 be the final step prior to interval I . For each cup j , the fractional amount of water in the cup at the beginning of interval I is

$$r_j + c_j([1, t_0]) \bmod 1. \quad (1)$$

Since r_j is uniformly random in $[0, 1]$, it follows that (1) is as well. The first $c_j(I) - \lfloor c_j(I) \rfloor$ units of water poured into cup j during interval I will therefore cross a threshold with probability exactly $c_j(I) - \lfloor c_j(I) \rfloor$. The next $\lfloor c_j(I) \rfloor$ units of water placed into cup j are then guaranteed to cause exactly $\lfloor c_j(I) \rfloor$ threshold crossings. The number of crossings in cup j during the step sequence is therefore $\lfloor c_j(I) \rfloor + X_j$, where X_j is 0-1 random variable with mean $c_j(I) - \lfloor c_j(I) \rfloor$, and where the randomness in X_j is due to the random initial offset r_j . Because r_1, r_2, \dots, r_n are independent, so are X_1, X_2, \dots, X_n . \square

One consequence of Lemma 4.1 is that, if a sequence of steps I has a large influence s , then each priority level ℓ will have at least $\Omega(s/q)$ cups that cross thresholds during interval I (recall that q is the number of priority levels).

Lemma 4.2 (The influence property, part 1). *Consider a sequence of steps I with influence s . Let ℓ be a priority level. With high probability*

²Note that, when counting the number of threshold crossings across a sequence of steps, the same threshold (j, i) may get crossed multiple times, and thus contribute more than 1 to the count.

in n , at least $\frac{s}{2q} - O(\log n)$ distinct cups with priority level ℓ cross thresholds during the sequence of steps I .

PROOF. By Lemma 4.1, the probability that cup j crosses at least one threshold during step sequence I is $\min(c_j(I), 1)$, independently of other cups j' . The number X of distinct cups that cross thresholds during interval I is therefore a sum of independent indicator random variables with mean s , where s is the influence of I . Since each cup has probability $\frac{1}{q}$ of having priority level ℓ , the number Y of cups with priority level ℓ that cross thresholds in interval I is a sum of independent indicator random variables with mean $\frac{s}{q}$.

If $s/q \leq O(\log n)$, the number of distinct cups with priority level ℓ to cross thresholds is at least $0 \geq \frac{s}{2q} - O(\log n)$ trivially. Suppose, on the other hand, that $s/q \geq c \log n$ for a sufficiently large constant c . Then by a Chernoff bound,

$$\Pr \left[Y < \frac{s}{2q} \right] \leq \exp \left[-\frac{s}{8q} \right] \leq \frac{1}{n^{c/8}},$$

completing the proof. \square

The proofs of the preceding lemmas have not needed to explicitly consider the effect of there being a potentially large number p of processors. In subsequent proofs, the multi-processor case will complicate the analysis in two ways. First, the emptier may sometimes dequeue a cup, even when there are more than p heavy thresholds in the queue (this can happen when the heavy thresholds all belong to a set of fewer than p cups). Second, and similarly, the emptier may sometimes be unable to remove a full p thresholds from the queue Q , even though $|Q| > p$ (this can happen if of the thresholds in Q belong to a set of fewer than p cups). It turns out that both of these problems can be circumvented using the fact that the emptying algorithm achieves small backlog. In particular, this ensures that no single cup can ever contribute more than $O(\log \log n + \log p)$ thresholds to Q :

Lemma 4.3 (K. [27]). *In any multi-processor cup game of poly n length, the asymmetric smoothed greedy algorithm achieves backlog $O(\log \log n + \log p)$ after each step, with high probability in n .*

PROOF. This follows from Theorem 5.1 of [27]. Although [27] considers the smoothed greedy algorithm (rather than the asymmetric smoothed greedy algorithm), the analysis applies without modification. \square

Using Lemma 4.3 as a tool to help in the case of $p > 1$, we now return to the analysis approach outlined in Section 3.

Remark 4.4. The proof of Lemma 4.3 given in [27] is highly non-trivial. We remark that, although Lemma 4.3 simplifies our analysis, there is also an alternative lighter weight approach that one can use in place of the lemma. In particular, one can begin by analyzing the *h -truncated cup game* for some sufficiently large $h \leq O(\log \log n + \log p)$. In this game, the height of each cup is deterministically bounded above by h , and whenever the height of a cup exceeds h , 1 unit of water is removed from the cup (and that unit does not count as part of the emptier's turn). The h -truncated cup game automatically satisfies the backlog property stated by Lemma 4.3, allowing for it to be analyzed without requiring the lemma. The analysis can then be used to bound the backlog of the h -truncated cup game to at most $h/2$ with high probability (using the analysis by

[27] of the greedy algorithm, applied to the $\tilde{O}(\log n + p)$ cups in the tail). It follows that with high probability, the h -truncated cup game and the (standard) cup game are indistinguishable. This means that the high-probability bounds on tail size for the h -truncated cup game carry over directly to the standard cup game.

The next lemma shows that, even though many threshold crossings may occur in a sequence of steps I , the size of the queue Q varies by only a small amount as a function of the influence of I .

Lemma 4.5 (The influence property, part 2). *Consider a sequence of steps I with influence at most s during a game of length at most poly n . For each step $t \in I$, let Q_t denote the queue after step t . With high probability in n ,*

$$||Q_{t_1}| - |Q_{t_2}|| \leq O(\sqrt{s \log n} + \log n + p(\log \log n + \log p))$$

for all subintervals $[t_1, t_2] \subseteq I$.

PROOF. We begin with a simpler claim:

Claim 4.6. *For any subinterval $I' \subseteq I$, the number of threshold crossings during I' is within $O(\sqrt{s \log n} + \log n)$ of $p|I'|$ with high probability in n .*

PROOF. Because I has influence at most s so does I' . Lemma 4.1 tells us that, during I' , the number of threshold crossings X that occur satisfies $\mathbb{E}[X] = p|I'|$. Moreover, X satisfies $X = A + \sum_{j=1}^n X_j$, where A is a fixed value and the X_j 's are independent 0-1 random variables, each taking value 1 with probability $c_j(I') - \lfloor c_j(I') \rfloor \leq \min(c_j(I'), 1)$. Note that I' has influence $\sum_j \min(c_j(I'), 1) \leq s$, and thus $\mathbb{E} \left[\sum_{j=1}^n X_j \right] \leq s$. By a multiplicative Chernoff bound, it follows that for $\delta < 1$,

$$\Pr[|X - \mathbb{E}[X]| \geq \delta s] \leq 2 \exp[-\delta^2 s/3].$$

Set $\delta = c\sqrt{\log n}/\sqrt{s}$ for a sufficiently large constant c . If $\delta > 1$, then $s \leq O(\log n)$ and $|X - \mathbb{E}[X]|$ is deterministically $O(\log n)$. Otherwise,

$$\Pr[|X - \mathbb{E}[X]| \geq c\sqrt{s \log n}] \leq 2 \exp[-c^2 \log n/3] \leq \frac{1}{\text{poly } n}.$$

Since $\mathbb{E}[X] = p|I'|$, it follows that the number of threshold crossings in interval I' is within $O(\sqrt{s \log n} + \log n)$ of $p|I'|$ with high probability in n . \square

Applying a union bound to the poly n subintervals of steps $I' \subseteq I$, Claim 4.6 tells us that every subinterval $I' \subseteq I$ contains $p|I'| \pm O(\sqrt{s \log n} + \log n)$ threshold crossings with high probability in n .

To complete the proof, consider some subinterval $I' \subseteq I$ and let m be the (absolute) amount by which Q changes in size during I' . We wish to show that $m \leq O(\sqrt{s \log n} + \log n + p(\log \log n + \log p))$.

Suppose that $|Q|$ shrinks by m during I' . Then the number of threshold crossings in subinterval I' would have to be at most $p|I'| - m$, meaning that $m \leq O(\sqrt{s \log n} + O(\log n))$, as desired.

Suppose, on the other hand, that $|Q|$ grows by m during I' . Call a step $t \in I'$ **removal-friendly** if the emptier removes p full units of water during step t (i.e., prior to the emptier removing water, there are at least p cups with height 1 or greater). By Lemma 4.3, with high probability in n , the size of Q after any removal-unfriendly step is at most $O(p(\log \log n + \log p))$. If I' consists exclusively of removal-friendly steps, then the filler must cross at least $p|I'| + m$ threshold

crossings in order to increase $|Q|$ by m ; thus $m \leq O(\sqrt{s \log n} + \log n)$. On the other hand, if I' contains at least one removal-unfriendly step, then there must be some last such step t in $I' = (t_0, t_1]$. Since $|Q| \leq O(p(\log \log n + \log p))$ after step t but $|Q| \geq m$ after step t_1 , it must be that during the steps $(t, t_1]$ the size of Q increases by at least $m - O(p(\log \log n + \log p))$. Since the interval (t, t_1) consists entirely of removal-friendly steps, we can apply the reasoning from the first case (i.e., the case of only removal-friendly steps) to deduce that $m \leq O(\sqrt{s \log n} + \log n + p(\log \log n + \log p))$, completing the proof. \square

Combined, Lemmas 4.2 and 4.5 give the influence property discussed in Section 3. Using this property, we can now relate the number of queued cups with priority level $\geq \ell$ to the number of queued cups with priority level $\geq \ell + 1$ for a given priority level $\ell \in \mathbb{N}$.

Lemma 4.7 (Accumulation of low-priority cups). *Let $t \leq \text{poly } n$, let K be the number of queued cups with priority level $\geq \ell$ after step t , and m be the number of queued cups with priority level $\geq \ell + 1$ after step t . With high probability in n ,*

$$m \leq O(\sqrt{qK \log n} + \log n + p(\log \log n + \log p)). \quad (2)$$

PROOF. For each $k \in \{1, 2, \dots, n\}$, define I_k to be the smallest step-interval ending at step t and with influence at least $2qk$ (or define $I_k = [1, t]$ if the total influence of $[1, t]$ is less than $2qk$). By Lemmas 4.2 and 4.5, each I_k satisfies the following two properties with high probability in n :

- **The many-crossings property.** Either I_k is all of $[1, t]$, or the number of priority-level- ℓ cups to cross thresholds during I_k is at least $k - O(\log n)$.
- **The low-variance property.** The size of Q varies by at most $B_k := O(\sqrt{qk \log n} + \log n + p(\log \log n + \log p))$ during I_k . To see this, we use the fact that I_k has influence at most $2qk + p$, which by Lemma 4.5 limits the amount by which Q varies to

$$O(\sqrt{(qk + p) \log n} + \log n + p(\log \log n + \log p)).$$

Since $\sqrt{(qk + p) \log n} \leq \sqrt{qk \log n} + \sqrt{p \log n} \leq \sqrt{qk \log n} + p + \log n$, it follows that the amount by which Q varies during I_k is at most $O(\sqrt{qk \log n} + \log n + p(\log \log n + \log p))$, with high probability in n .

Collectively, this pair of properties is called the **influence property**. By a union bound, the influence property holds for all $k \in \{1, 2, \dots, n\}$ with high probability in n . It follows that the property also holds for $k = K$ (recall K is the number of queued cups with priority level $\geq \ell$ after step t).

If $I_K = [1, t]$, then the total size of Q can be at most B_K (since Q begins as size 0 at the start of I_K). It follows that $m \leq B_K$, meaning that (2) is immediate. In the rest of the proof, we focus on the case in which $I_K \neq [1, t]$.

If the emptier never dequeues any priority-level- ℓ cups during I_K , then by the many-crossings property, there are at least $K - O(\log n)$ priority-level- ℓ cups queued after step t . The number of queued cups with priority levels greater than ℓ is therefore at most $O(\log n)$, as desired.

Suppose, on the other hand, that there is at least one step in I_K at which the emptier dequeues a priority-level- ℓ (or smaller) cup, and let t^* be the last such step. Let Q_{t^*} be the set of queued thresholds after t^* , and let Q_t be the set of queued thresholds after step t . Call a threshold in Q_{t^*} **permanent** if it is a light threshold for a cup with priority level $\leq \ell$. All permanent thresholds in Q_{t^*} are guaranteed to also be in Q_t , since t^* is the final step in I_K during which the emptier dequeues such a threshold. The non-permanent thresholds in Q_{t^*} must reside in a set of fewer than p cups, since the emptier would rather have dequeued one of them during step t^* than to have dequeued a cup with priority level $\leq \ell$. By Lemma 4.3, the number of non-permanent thresholds in Q_{t^*} is therefore at most $O(p \log \log n + p \log p)$, with high probability in n .

By the low-variance property, the sizes of Q_t and Q_{t^*} differ by at most B_K . It follows that the permanent thresholds in Q_{t^*} make up all but

$$B_K + O(p \log \log n + p \log p) \leq O(B_K)$$

of the thresholds in Q_t .

Recall that Q_t contains thresholds from K different cups with priority level $\geq \ell$. It follows that Q_{t^*} contains permanent thresholds from at least $K - O(B_K)$ different cups with priority level $\geq \ell$. The permanent thresholds in Q_{t^*} are all for cups with priority level $\leq \ell$, however. Thus there are at least $K - O(B_K)$ cups with priority level ℓ that are queued after step t^* and remain queued after step t . This bounds the number of queued cups after step t with priority level greater than ℓ by at most $O(B_K)$, completing the proof. \square

Since the number of queued cups with priority level ≥ 1 can never exceed n , Lemma 4.7 allows for us to bound the number of queued cups with priority level $\geq \ell$ inductively. We argue that if q is a sufficiently large constant multiple of $\log \log n$, then the number of queued cups with priority level $\geq q/2$ never exceeds $O(\log n \log \log n + p(\log \log n + \log p))$, with high probability in n . This can then be used to obtain $(\log n \log \log n + p \log p)$ -unpredictability, as defined in Section 3.

Lemma 4.8 (The unpredictability guarantee). *Consider a cup game of length $\text{poly } n$. For any step t , and for any set of cups S whose size is a sufficiently large constant multiple of $\log n \log \log n + p \log p$, at least one cup in S is not queued after step t , with high probability in n . In other words, each step t in the game satisfies $(\log n \log \log n + p \log p)$ -unpredictability.*

PROOF. Suppose the number of priority levels q is set to be a sufficiently large constant multiple of $\log \log n$. For $\ell \in \{1, 2, \dots, q\}$, let m_ℓ denote the maximum number of queued cups with priority level $\geq \ell$ during the game. We claim that $m_{q/2} \leq O(\log n \log \log n + p \log p)$ with high probability in n .

By Lemma 4.7, for any $1 \leq \ell < q$,

$$m_{\ell+1} \leq O(\sqrt{q m_\ell \log n} + \log n + p \log \log n + p \log p),$$

with high probability in n . If we define $X = q \log n + \log n + p \log \log n + p \log p$, then it follows that,

$$m_{\ell+1} \leq O(\sqrt{X m_\ell} + X). \quad (3)$$

For each priority level ℓ , let δ_ℓ be the ratio $\delta_\ell = \frac{m_\ell}{X}$. By (3), for any $1 \leq \ell < q$, either $\delta_{\ell+1} \leq O(1)$ or

$$\delta_{\ell+1} \leq O(\sqrt{\delta_\ell}).$$

It follows that, as long as q is a sufficiently large constant multiple of $\log \log n$, then $\delta_{q/2} \leq O(1)$, and thus $m_{q/2} \leq O(X)$.

Now consider a set of cups S of size at least cX , where c is a sufficiently large constant. By a Chernoff bound, the number of cups with priority level greater than $q/2$ in S is at least $cX/4$, with high probability in n . Since c is a sufficiently large constant, and since $m_{q/2} \leq O(X)$, this implies that S contains more than $m_{q/2}$ cups with priority level $q/2$ or greater. Thus the priority-level- ℓ cups in S cannot all be queued after step t .

Note that $X \leq O(\log n \log \log n + p \log p)$. Thus every set S whose size is a sufficiently large constant multiple of $\log n \log \log n + p \log p$ has high probability of containing at least one non-queued cup after step t , completing the proof of $\log n \log \log n + p \log p$ -unpredictability. \square

To complete the analysis of the algorithm, we must formalize the connection between the unpredictability guarantee and tail size.

Lemma 4.9. *Suppose that a (randomized) emptying algorithm for the p -processor cup game on n cups satisfies R -unpredictability in the steps of any game of polynomial length, and further satisfies the “greediness property” that whenever there is a cup of height at least 2, the algorithm empties out of such a cup. Then in any game of polynomial length, the tail size s after each step t is $O(R + p)$ with high probability in n .*

PROOF. Consider a polynomial $f \in \text{poly } n$, let c be a large constant, and let $t \leq \text{poly } n$. Suppose for contradiction that there is a filling strategy such that, at time t the tail size is at least $cR + p$ with probability at least $1/f(t)$. Since the tail size is at least $cR + p$ at time t , then during each of steps $I = \{t + 1, \dots, t + \lceil cR/p \rceil\}$, the emptier removes water exclusively from cups with fills at least 2. This means that the set of cups containing 1 or more units of water is monotonically increasing during steps $t + 1, \dots, t + \lceil cR/p \rceil$. If the filler places 1 unit of water into each cup $1, 2, \dots, cR$ during steps $t + 1, \dots, t + \lceil cR/p \rceil$, then it follows that each of cups $1, 2, \dots, cR$ has fill 1 or greater after step $t + \lceil cR/p \rceil$.

The preceding construction guarantees that, with probability at least $1/f(t)$, all of cups $1, 2, \dots, cR$ contain at least 1 unit of water at step $t + \lceil cR/p \rceil$. If c is a sufficiently large constant, this violates R -unpredictability at step $t + \lceil cR/p \rceil$, a contradiction. \square

Using the unpredictability guarantee, we can now complete the algorithm analysis:

Theorem 4.10. *Consider a p -processor cup game that lasts for poly n steps and in which the emptier follows the asymmetric smoothed greedy algorithm. Then with high probability in n , the number of cups containing 2 or more units of water never exceeds $O(\log n \log \log n + p \log p)$ and the backlog never exceeds $O(\log \log n + \log p)$ during the game.*

PROOF. By Lemma 4.8, $(\log n \log \log n + p \log p)$ -unpredictability holds for each step in any game of length poly n . By Lemma 4.9, it follows that the tail size remains at most $O(\log n \log \log n + p \log p)$, with high probability in n , during any game of length poly n .

Lemma 4.3 bounds the height of the fullest cup in each step by $O(\log \log n + \log p)$ with high probability in n . Alternatively, by considering a cup game consisting only of the cups that contain greater than 2 units of water, the analysis of the deterministic greedy emptying algorithm (see [1] for $p = 1$ and [27] for $p > 1$) on $O(\log n \log \log n + p \log p)$ cups implies that no cup ever contains more than $O(\log \log n + \log p)$ water, with high probability in n . \square

5 LOWER BOUNDS

In this section we prove that the asymmetric smoothed greedy algorithm achieves (near) optimal tail size within the class of backlog-bounded algorithms.

An emptying algorithm is **backlog bounded** if the algorithm guarantees that the backlog never exceeds $f(n)$ for some polynomial f . This is a weak requirement in that the greedy algorithm achieves a bound of $O(\log n)$ on backlog [1, 27]. The main result in this section states that any backlog-bounded emptying algorithm must allow for a tail size of $\tilde{\Omega}(\log n + p)$ with probability $\frac{1}{\text{poly } n}$. The lower bound continues to hold, even when the height-requirement for a cup to be in the tail is increased to an arbitrarily large constant (rather than 2). When $p = 1$, the lower bound also applies to non-backlog-bounded emptying algorithms (Lemma 5.3).

Theorem 5.1. *Let c_1 be a constant, and suppose $n \geq p + c_2$ for sufficiently large constant c_2 . For any backlog-bounded emptying strategy, there is a poly n -step oblivious randomized filling strategy that gives the following guarantee. After the final step of the filling strategy, there are at least $\Omega(\log n / \log \log n + p)$ cups with fill c_1 or greater, with probability at least $\frac{1}{\text{poly } n}$.*

To prove Theorem 5.1, we begin by describing a simple lower-bound construction that we call the (p, k, c) -filling strategy. The strategy is structurally similar to the lower-bound construction for backlog given by Bender et al [11].

Lemma 5.2. *Let $k, c \in \mathbb{N}$ such that $c \geq 2$, $k \leq n$, and $\frac{k}{pe^c} \geq 2$. Then there exists an $O(k)$ -step oblivious randomized filling strategy for the p -processor cup game on n cups that causes $\Omega(k/e^c)$ cups to each have fill at least $\Theta(c)$, with probability at least $\frac{1}{k^k}$. We call this strategy the (p, k, c) -filling strategy.*

PROOF. Define the (p, k, c) -filling strategy for the p -processor cup game on $n \geq k$ cups as follows. In each step i of the strategy, the filler places $\frac{p}{k-p(i-1)}$ units of water into each of $k - p(i-1)$ cups. The sets of cups S_i used in each step i are selected so that $S_{i+1} = S_i \setminus \{x_1, x_2, \dots, x_p\}$ for some random distinct $x_1, x_2, \dots, x_p \in S_i$. The (p, k, c) -filling strategy completes after t steps where $t = \lfloor \frac{k}{p} (1 - e^{-c}) \rfloor - 1$. Note that $k - pi \geq p$ for every step i .

We say that the (p, k, c) -filling strategy **succeeds** if at the beginning of each step i none of the cups in S_i have been touched (i.e., emptied from) by the emptier. If the (p, k, c) -filling strategy succeeds, then at the end the i -th step of the strategy there will be

$k - pi$ cups each with fill

$$\begin{aligned} & \frac{p}{k} + \frac{p}{k-p} + \cdots + \frac{p}{k-p(i-1)} \\ &= \Theta\left(\frac{1}{k} + \frac{1}{k-1} + \cdots + \frac{1}{k-pi+1}\right) \\ &= \Theta\left(\log \frac{k}{k-pi}\right), \end{aligned}$$

where the first equality uses the fact that $k - pi \geq p$.

Now consider the final step t of a successful (p, k, c) -filling strategy. By the requirement that $\frac{k}{pe^c} \geq 2$,

$$k - pt = k - p \left\lfloor \frac{k}{p} (1 - 1/e^c) \right\rfloor - p \geq \frac{k}{e^c} - p \geq \frac{k}{2e^c}.$$

It follows that, after step t of a successful (p, k, c) -filling strategy, there are at least $\Theta(k/e^c)$ cups, each with fill at least

$$\Omega\left(\log \frac{k}{k/(2e^c)}\right) = \Omega(c).$$

Next we evaluate the probability of a (p, k, c) -filling strategy being successful. If the first i steps of the (p, k, c) -filling strategy all succeed, then the $(i+1)$ -th step has probability at least $\frac{1}{k^p}$ of succeeding. In particular, the emptier may touch up to p cups $j_1, \dots, j_p \in S_i$ during step i , and then the set $S_{i+1} = S_i \setminus \{x_1, \dots, x_p\}$ has probability at least $\frac{1}{k^p}$ of removing a superset of those cups from S_i to get S_{i+1} . Since there are at most k/p steps, the (p, k, c) -filling strategy succeeds with probability at least $\frac{1}{k^k}$. \square

If we assume that $\log n / \log \log n$ is a sufficiently large constant multiple of p , then we can apply Lemma 5.2 directly to achieve a tail size of size $\tilde{\Omega}(\log n)$. (Furthermore, note that Lemma 5.3 does not have any requirement that the emptier be backlog-bounded.)

Lemma 5.3. *Let c_1 be a positive constant, and suppose $p \leq \frac{\log n}{c_2 \log \log n}$ for a sufficiently large constant c_2 (where c_2 is large relative to c_1). Then there is an $O(\log n / \log \log n)$ -step oblivious randomized filling strategy for the p -processor cup game on n cups that causes $\Omega(\log n)$ cups to all have height c_1 or greater after some step $t \leq \text{poly } n$ with probability at least $\frac{1}{\text{poly } n}$.*

PROOF. Let $c \in \mathbb{N}$ be a sufficiently large constant compared to c_1 . By assumption that c_2 is sufficiently large in terms of c_1 , we may also assume that

$$\frac{\log n / \log \log n}{pe^c} \geq 2. \quad (4)$$

By (4), we can use Lemma 5.2 to analyze the $(p, \log n / \log \log n, c)$ -filling strategy. The strategy causes

$$\Omega\left(\frac{\log n}{\log \log n} \cdot e^{-c}\right) \geq \Omega(\log n / \log \log n)$$

cups to all have height at least c_1 with probability at least

$$(\log n / \log \log n)^{-\log n / \log \log n} \geq \frac{1}{\text{poly } n}.$$

\square

The next lemma gives a filling strategy for achieving tail size $\Omega(p)$ against any backlog-bounded emptying strategy. Remarkably, the construction in Lemma 5.4 succeeds with probability $1 - e^{-\Omega(p)}$ (rather than with probability $1/\text{poly } n$).

Lemma 5.4. *Let c_1 be a constant, and suppose $n \geq p + c_2$ for sufficiently large constant c_2 . For any backlog-bounded emptying strategy, there is a poly n -step oblivious randomized filling strategy that gives the following guarantee. After the final step of the filling strategy, there are at least $\Omega(p)$ cups with fill c_1 or greater, with probability $1 - e^{-\Omega(p)}$.*

PROOF. For the sake of simplicity, we allow for the filler to sometimes **swap** two cups, meaning that the labels of the cups are interchanged.

The basic building block of the algorithm is a **mini-phase**, which consists of $O(1)$ steps. In each step of a mini-phase the filler places 1 unit of water into each of cups $1, 2, \dots, p-1$, and then strategically distributes 1 additional unit of water among cups $p, p+1, \dots, n$. Using the final unit of water, the filler follows a $(1, ce^c, c)$ -filling strategy on cups $p, p+1, \dots, n$, where c is a sufficiently large constant relative to c_1 satisfying $n \geq p + ce^c$. We say that a mini-phase **succeeds** if the emptier removes only 1 unit of water from cups $\{p, p+1, \dots, n\}$ during each step in the mini-phase, and the $(1, ce^c, c)$ -filling strategy succeeds within the mini-phase. By the Lemma 5.2, any successful mini-phase will cause at least one cup j to have fill at least c_1 at the end of the mini-phase (and the filler will know j).

Mini-phases are composed together by the filler to get **phases**. During each i -th phase, the filler selects a random $w_i \in [1, f(n)n^2]$ and performs w_i mini-phases (recall that $f(n)$ is the polynomial such that the emptier achieves backlog $f(n)$ or smaller). After the w_i -th mini-phase, the filler swaps cups i and j , where i is the phase number and j is the cup containing fill $\geq c_1$ in the event that the most recent mini-phase succeeded. The full filling algorithm consists of $p-1$ phases.

We claim that each phase i has constant probability of ending in a successful mini-phase (and thus swapping cup i with a new cup $j \geq p$ having fill $\geq c_1$). Using this claim, one can complete the analysis as follows. If the swap in phase i is at the end of a successful mini-phase, then after the swap, the (new) cup i will have fill $\geq c_1$, and will continue to have fill $\geq c_1$ for the rest of the filling algorithm, since the filler puts 1 unit in cup i during every remaining step. At the end of the algorithm, the number of cups with fill $\geq c_1$ is therefore bounded below by a sum of $p-1$ independent 0-1 random variables with total mean $\Omega(p)$. This means that the number of such cups with fill $\geq c_1$ is at least $\Omega(p)$ with probability $1 - e^{-\Omega(p)}$, as desired.

It remains to analyze the probability that a given phase i ends with a successful mini-phase.

Call a mini-phase **clean** if the emptier removes 1 unit of water from each cup $1, 2, \dots, p-1$ during each step of the mini-phase, and **dirty** otherwise. Because each dirty mini-phase increases the total amount of water in cups $1, 2, \dots, p-1$ by at least 1, and because the emptying algorithm prevents backlog from ever exceeding $f(n)$, there can be at most $O(pf(n))$ dirty mini-phases during phase i .

By Lemma 5.2, each mini-phase (independently) has at least a constant probability of either being dirty or of succeeding. Out of the $f(n)n^2$ possible mini-phases in phase i , there can only be $O(f(n)p) \leq o(f(n)n^2)$ dirty mini-phases. It follows that, with probability $1 - e^{-\Omega(f(n)n^2)}$, at least a constant fraction of the possible mini-phases s succeed (or would have succeeded in the event that

w_i were at least as large as s). Thus the w_i -th mini-phase succeeds with constant probability. \square

Combining the preceding lemmas, we prove Theorem 5.1.

PROOF OF THEOREM 5.1. If $p \geq \Omega(\log n / \log \log n)$, then the theorem follows immediately from Lemma 5.4. On the other hand, if p is a sufficiently large constant factor smaller than $\log n / \log \log n$, then the theorem follows from Lemma 5.3. \square

6 LOWER BOUNDS AGAINST UNENDING GUARANTEES

In this section, we give upper and lower bounds for **unending guarantees**, which are probabilistic guarantees that hold for each step t , even when t is arbitrarily large. As a convention, we will use $f_j(t)$ to denote the fill of cup j after step t .

The main result of the section is a lower bound showing that no **monotone stateless emptier** can achieve an unending guarantee of $o(\log n)$ backlog.

Definition 6.1. An emptier is said to be **stateless** if the emptier's decision depends only on the state of the cups at each step. An emptier is said to be **monotone** if the following holds: given a state S of the cups in which the emptier selects some cup j to empty, if we define S' to be S except that the amount of water in some cup $i \neq j$ has been reduced, then the emptier still selects cup j in state S' . A **monotone stateless emptier** is any emptier that is both monotone and stateless.

The monotonicity and stateless property dictate only how the emptier selects a cup j in each step. Once a cup j is selected the emptier is permitted to either (a) remove 1 full unit of water from that cup, or (b) skip their turn. This decision is allowed to be an arbitrary function of the state of the cups.

We begin in Section 6.1 by showing that all monotone stateless emptiers can be modeled as using a certain type of **score function** to make emptying decisions.

In Section 6.2, we give an oblivious filling strategy, called the **fuzzing algorithm**, that prevents monotone stateless emptiers from achieving unending probabilistic guarantees of $o(\log n)$ backlog (in fact, the filling strategy places an expected $\Theta(n^{2/3} \log n)$ water into $\Theta(n^{2/3})$ cups, meaning that bounds on tail size are also not viable, unless backlog is allowed to be polynomially large). The fuzzing algorithm is named after what is known as the **fuzzing technique** [36] for detecting security vulnerabilities in computer systems – by barraging the system with random noise, one accidentally discovers and exploits the structural holes of the system.

In Section 6.3 we show that the fuzzing algorithm continues to prevent unending guarantees, even when the emptier is equipped with a global clock, allowing for the emptier to adapt to the number of steps that have occurred so far in the game.

Finally, in Section 6.4, we determine the exact values of the resource-augmentation parameter ε for which the smoothed greedy and asymmetric smoothed greedy emptying algorithms achieve single-processor unending guarantees. In particular, we show that the minimum attainable value of ε is $2^{-\text{polylog } n}$.

For the sake of brevity, we defer the proofs of the results in this section to the extended version of the paper, and we limit ourselves here to the formal statements and discussion of the results.

6.1 Score-Based Emptiers

In this section, we give an equivalence between monotone stateless emptiers, and what we call score-based emptiers.

A **score-based emptier** has **score functions** $\sigma_1, \sigma_2, \dots, \sigma_n$. When selecting which cup to empty from, the emptier selects the cup j whose fill f_j maximizes $\sigma_j(f_j)$. The emptier can then select whether to either (a) remove 1 full unit of water from the cup, or (b) skip their turn; this decision is an arbitrary function of the state of the cups. The score functions are required to be monotonically increasing functions, meaning that $\sigma_i(a) < \sigma_i(b)$ whenever $a < b$. Moreover, in order to break ties, all of the scores in the multiset $\{\sigma_i(j/2) \mid i \in [n], j \in \mathbb{Z}^+\}$ are required to be distinct. (We only consider fills of the form $j/2$ because in our lower bound constructions all fills will be multiples of $1/2$.)

It is easy to see that any score-based emptier is also a monotone stateless emptier. The following theorem establishes that the other direction is true as well:

Theorem 6.2. Consider cup games in which the filler always places water into cups in multiples of $1/2$. For these cup games, every monotone stateless emptying algorithm is equivalent to some score-based emptying algorithm.

6.2 The Oblivious Fuzzing Filling Algorithm

In this section, we describe a simple filling algorithm that, when pitted against a score-based emptier, achieves backlog $\Omega(\log n)$ after $n^{\Theta(n \log n)}$ steps with at least constant probability. Note that, throughout this section, we focus only on cup games that do *not* have resource augmentation.

The filling strategy, which we call the **oblivious fuzzing algorithm** has a very simple structure. At the beginning of the algorithm, the filler randomly permutes the labels $1, 2, \dots, n$ of the cups. The filler then begins their strategy by spending a large number (i.e., $n^{\Theta(n \log n)}$) of steps randomly placing water into cups $1, 2, \dots, n$. The filler then disregards cup n (note that cup n is a random cup due to the random-permutation step!), and spends a large number of steps randomly placing water into cups $1, 2, \dots, n-1$. The filler then disregards cup $n-1$ and spends a large number of steps randomly placing water into cups $1, 2, \dots, n-2$, and so on.

Formally, the oblivious fuzzing algorithm works as follows. Let $c \in \mathbb{N}$ be a sufficiently large constant, and relabel the cups (from the filler's perspective) with a random permutation of $1, 2, \dots, n$. The filling strategy consists of n phases of n^c steps. The i -th phase is called the $(n-i+1)$ -**cup phase** because it focuses on cups $1, 2, \dots, (n-i+1)$. In each step of the i -th phase, the filler selects random values $x_1, x_2 \in \{1, 2, \dots, n-i+1\}$ uniformly and independently, and then places $\frac{1}{2}$ water into each of cups x_1, x_2 . If $x_1 = x_2$, then the cup x_1 receives a full unit of water.

One interesting characteristic of the oblivious fuzzing algorithm is that it represents a natural workload in the scheduling problem that the cup game models. One can think of the cups as representing n tasks and water as representing work that needs to be scheduled. In this scheduling problem, the oblivious fuzzing filling algorithm simply assigns work to tasks at random, and selects one task every $n^{cn \log n}$ steps to stop receiving new work.

In the extended version of this paper, we prove the following theorem.

Theorem 6.3. *Consider a cup game on n cups. Suppose that the emptier follows a score-based emptying algorithm, and that the filler follows the oblivious fuzzing filling algorithm. Then at the beginning of the $n^{2/3}$ -cup phase, the average fill of cups $1, 2, \dots, n^{2/3}$ is $\Omega(\log n)$, in expectation.*

6.3 Giving the Emptier a Time Stamp

In this section, we establish that unending guarantees continue to be impossible, even if the score-based emptier is permitted to change their algorithm based on a global time stamp.

A **dynamic score-based emptying algorithm** \mathcal{A} is dictated by a sequence $\langle X_1, X_2, X_3, \dots \rangle$, where each X_i is a score-based emptying algorithm. On step t of the cup game, the algorithm \mathcal{A} follows algorithm X_t .

Define the **extended oblivious fuzzing filling algorithm** to be the oblivious fuzzing filling strategy, except that each phase's length is increased to consist of $T(n)$ steps, where T is a sufficiently large function of n .

Theorem 6.4. *Consider a cup game on n cups. Suppose the emptier is a dynamic score-based emptier. Suppose the filler follows the extended oblivious fuzzing filling algorithm. Then at the beginning of the $n^{2/3}$ -cup phase, the average fill of cups $1, 2, \dots, n^{2/3}$ is $\Omega(\log n)$, in expectation.*

6.4 Unending Guarantees with Small Resource Augmentation

In this section we establish that, even though resource augmentation $\varepsilon > 0$ is needed to achieve unending guarantees for the smoothed greedy (and asymmetric smoothed greedy) emptying algorithms, the amount of resource augmentation that is necessary is substantially smaller than was previously known. In particular, we give unending guarantees when $\varepsilon = 2^{-\text{polylog } n}$.

Theorem 6.5 states an unending guarantee for the smoothed greedy emptying algorithm, using $\varepsilon = 2^{-\text{polylog } n}$.

Theorem 6.5. *Consider a single-processor cup game in which the emptier follows the smoothed greedy emptying algorithm, and the filler is an oblivious filler. If the game has resource augmentation parameter $\varepsilon \geq 2^{-\text{polylog } n}$, then each step t achieves backlog $O(\log \log n)$ with probability $1 - 2^{-\text{polylog } n}$ (where the exponent in the polylog is a constant of our choice).*

Theorem 6.6 states an unending guarantee for the asymmetric smoothed greedy emptying algorithm, using $\varepsilon = 2^{-\text{polylog } n}$.

Theorem 6.6. *Consider a single-processor cup game in which the emptier follows the asymmetric smoothed greedy emptying algorithm, and the filler is an oblivious filler. If the game has resource augmentation parameter $\varepsilon \geq 2^{-\text{polylog } n}$, then each step t achieves tail size $O(\text{polylog } n)$ and the backlog $O(\log \log n)$ with probability $1 - 2^{-\text{polylog } n}$ (where the exponent in the polylog in the probability is a constant of our choice).*

Whereas Theorems 6.5 and 6.6 give unending guarantees for the smoothed greedy (and asymmetric smoothed greedy) emptying algorithms using resource augmentation $\varepsilon = 1/2^{\text{polylog } n}$, Theorem 6.7 shows that such guarantees cannot be achieved with smaller resource augmentation.

Theorem 6.7. *Consider a single-processor cup game on n cups. Suppose $\varepsilon = 1/2^{\log^{\omega(1)} n}$, and suppose the emptier follows either the smoothed greedy emptying algorithm or the asymmetric smoothed greedy emptying algorithm. Then there is an oblivious filling strategy that causes there to be a step t at which the expected backlog is $\omega(\log \log n)$.*

REFERENCES

- [1] Micah Adler, Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul Goldberg, and Mike Paterson. 2003. A Proportionate Fair Scheduling Rule with Good Worst-case Performance. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. 101–108. <https://doi.org/10.1145/777412.777430>
- [2] Amihoud Amir, Martin Farach, Ramana M. Idury, Johannes A. La Poutré, and Alejandro A. Schäffer. 1995. Improved Dynamic Dictionary Matching. *Inf. Comput.* 119, 2 (1995), 258–282. <https://doi.org/10.1006/inco.1995.1090>
- [3] Amihoud Amir, Gianni Franceschini, Roberto Grossi, Tsvi Kopelowitz, Moshe Lewenstein, and Noa Lewenstein. 2014. Managing Unbounded-Length Keys in Comparison-Driven Data Structures with Applications to Online Indexing. *SIAM J. Comput.* 43, 4 (2014), 1396–1416. <https://doi.org/10.1137/110836377>
- [4] Yossi Azar and Arik Litichevsky. 2006. Maximizing Throughput in Multi-Queue Switches. *Algorithmica* 45, 1 (2006), 69–90. <https://doi.org/10.1007/s00453-005-1190-x>
- [5] Amotz Bar-Noy, Ari Freund, Shimon Landa, and Joseph Naor. 2003. Competitive On-Line Switching Policies. *Algorithmica* 36, 3 (2003), 225–247. <https://doi.org/10.1007/s00453-003-1014-9>
- [6] Amotz Bar-Noy, Aviv Nisgav, and Boaz Patt-Shamir. 2002. Nearly optimal perfectly periodic schedules. *Distributed Comput.* 15, 4 (2002), 207–220. <https://doi.org/10.1007/s00446-002-0085-1>
- [7] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. 1996. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica* 15, 6 (01 Jun 1996), 600–625. <https://doi.org/10.1007/BF01940883>
- [8] Sanjoy K. Baruah, Johannes Gehrke, and C. Greg Plaxton. 1995. Fast scheduling of periodic tasks on multiple resources. In *Proceedings of IPSP '95, The 9th International Parallel Processing Symposium, April 25–28, 1995, Santa Barbara, California, USA*. IEEE Computer Society, 280–288. <https://doi.org/10.1109/IPSP.1995.395946>
- [9] Michael A. Bender, Jake Christensen, Alex Conway, Martin Farach-Colton, Rob Johnson, and Meng-Tsung Tsai. 2019. Optimal Ball Recycling. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6–9, 2019*, Timothy M. Chan (Ed.). SIAM, 2527–2546. <https://doi.org/10.1137/1.9781611975482.155>
- [10] Michael A. Bender, Rathish Das, Martin Farach-Colton, Rob Johnson, and William Kuszmaul. 2020. Flushing Without Cascades. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5–8, 2020*, Shuchi Chawla (Ed.). SIAM, 650–669. <https://doi.org/10.1137/1.9781611975994.40>
- [11] Michael A. Bender, Martin Farach-Colton, and William Kuszmaul. 2019. Achieving optimal backlog in multi-processor cup games. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23–26, 2019*, Moses Charikar and Edith Cohen (Eds.). ACM, 1148–1157. <https://doi.org/10.1145/3313276.3316342>
- [12] Michael A. Bender, Sándor P. Fekete, Alexander Kröller, Vincenzo Liberatore, Joseph S. B. Mitchell, Valentin Polishchuk, and Jukka Suomela. 2015. The minimum backlog problem. *Theor. Comput. Sci.* 605 (2015), 51–61. <https://doi.org/10.1016/j.tcs.2015.08.027>
- [13] Marijke Hans L. Bodlaender, Cor A. J. Hurkens, Vincent J. J. Kusters, Frank Staals, Gerhard J. Woeginger, and Hans Zantema. 2012. Cinderella versus the Wicked Stepmother. In *IFIP International Conference on Theoretical Computer Science*. 57–71.
- [14] Marijke H. L. Bodlaender, Cor A. J. Hurkens, and Gerhard J. Woeginger. 2011. The Cinderella Game on Holes and Anti-holes. In *Proceedings of the 37th International Conference on Graph-Theoretic Concepts in Computer Science (WG)*. 71–82. https://doi.org/10.1007/978-3-642-25870-1_8
- [15] Marek Chrobak, János Csirik, Csanád Imreh, John Noga, Jiri Sgall, and Gerhard J. Woeginger. 2001. The Buffer Minimization Problem for Multiprocessor Scheduling with Conflicts. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP)*, Vol. 2076. 862–874. https://doi.org/10.1007/3-540-48224-5_70
- [16] Peter Damaschke and Zhen Zhou. 2005. On queuing lengths in on-line switching. *Theor. Comput. Sci.* 339, 2–3 (2005), 333–343. <https://doi.org/10.1016/j.tcs.2005.03.025>
- [17] Paul Dietz and Daniel Sleator. 1987. Two Algorithms for Maintaining Order in a List. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC)* (New York, New York, USA). 365–372. <https://doi.org/10.1145/283950.283950>

- 1145/28395.28434
- [18] Paul F. Dietz and Rajeev Raman. 1991. Persistence, Amortization and Randomization. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 78–88. <http://dl.acm.org/citation.cfm?id=127787.127809>
 - [19] Johannes Fischer and Pawel Gawrychowski. 2015. Alphabet-Dependent String Searching with Wexponential Search Trees. In *Combinatorial Pattern Matching - 26th Annual Symposium, CPM 2015, Ischia Island, Italy, June 29 - July 1, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9133)*, Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro (Eds.). Springer, 160–171. https://doi.org/10.1007/978-3-319-19929-0_14
 - [20] Rudolf Fleischer and Hisashi Koga. 2004. Balanced Scheduling toward Loss-Free Packet Queuing and Delay Fairness. *Algorithmica* 38, 2 (01 Feb 2004), 363–376. <https://doi.org/10.1007/s00453-003-1064-z>
 - [21] H. Richard Gail, George A. Grover, Roch Guérin, Sidney L. Hantler, Zvi Rosberg, and Moshe Sidi. 1993. Buffer Size Requirements Under Longest Queue First. *Perform. Evaluation* 18, 2 (1993), 133–140. [https://doi.org/10.1016/0166-5316\(93\)90033-Q](https://doi.org/10.1016/0166-5316(93)90033-Q)
 - [22] Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. 2017. Bamboo Garden Trimming Problem (Perpetual Maintenance of Machines with Different Attendance Urgency Factors). In *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10139)*, Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria (Eds.). Springer, 229–240. https://doi.org/10.1007/978-3-319-51963-0_18
 - [23] Michael H. Goldwasser. 2010. A survey of buffer management policies for packet switches. *SIGACT News* 41, 1 (2010), 100–128. <https://doi.org/10.1145/1753171.1753195>
 - [24] Michael T. Goodrich and Pawel Pszona. 2013. Streamed Graph Drawing and the File Maintenance Problem. In *Graph Drawing - 21st International Symposium, GD 2013, Bordeaux, France, September 23-25, 2013, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8242)*, Stephen K. Wismath and Alexander Wolff (Eds.). Springer, 256–267. https://doi.org/10.1007/978-3-319-03841-4_23
 - [25] Nan Guan and Wang Yi. 2012. Fixed-Priority Multiprocessor Scheduling: Critical Instant, Response Time and Utilization Bound. In *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012, Shanghai, China, May 21-25, 2012*. IEEE Computer Society, 2470–2473. <https://doi.org/10.1109/IPDPSW.2012.305>
 - [26] Tsvi Kopelowitz. 2012. On-Line Indexing for General Alphabets via Predecessor Queries on Subsets of an Ordered List. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*. IEEE Computer Society, 283–292. <https://doi.org/10.1109/FOCS.2012.79>
 - [27] William Kuszmaul. 2020. Achieving Optimal Backlog in the Vanilla Multi-Processor Cup Game. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, Shuchi Chawla (Ed.). SIAM, 1558–1577. <https://doi.org/10.1137/1.9781611975994.96>
 - [28] Ami Litman and Shiri Moran-Schein. 2005. On distributed smooth scheduling. In *SPAA 2005: Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, July 18-20, 2005, Las Vegas, Nevada, USA*, Phillip B. Gibbons and Paul G. Spirakis (Eds.). ACM, 76–85. <https://doi.org/10.1145/1073970.1073982>
 - [29] Ami Litman and Shiri Moran-Schein. 2009. Smooth Scheduling Under Variable Rates or the Analog-Digital Confinement Game. *Theor. Comp. Sys.* 45, 2 (June 2009), 325–354. <https://doi.org/10.1007/s00224-008-9134-x>
 - [30] Ami Litman and Shiri Moran-Schein. 2011. On Centralized Smooth Scheduling. *Algorithmica* 60, 2 (2011), 464–480. <https://doi.org/10.1007/s00453-009-9360-x>
 - [31] Chung Laung Liu. 1969. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary*, 1969 (1969).
 - [32] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.
 - [33] Mark Moir and Srikanth Ramamurthy. 1999. Pfair Scheduling of Fixed and Migrating Periodic Tasks on Multiple Resources. In *Proceedings of the 20th IEEE Real-Time Systems Symposium, Phoenix, AZ, USA, December 1-3, 1999*. IEEE Computer Society, 294–303. <https://doi.org/10.1109/REAL.1999.818857>
 - [34] Christian Worm Mortensen. 2003. Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*. ACM/SIAM, 618–627. <http://dl.acm.org/citation.cfm?id=644108.644210>
 - [35] Michael Rosenblum, Michel X. Goemans, and Vahid Tarokh. 2004. Universal Bounds on Buffer Size for Packetizing Fluid Policies in Input Queued, Crossbar Switches. In *Proceedings IEEE INFOCOM 2004, The 23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China, March 7-11, 2004*. IEEE, 1126–1134. <https://doi.org/10.1109/INFCOM.2004.1356999>
 - [36] Michael Sutton, Adam Greene, and Pedram Amini. 2007. *Fuzzing: brute force vulnerability discovery*. Pearson Education.
 - [37] Berthold Vöcking. 2003. How asymmetry helps load balancing. *J. ACM* 50, 4 (2003), 568–589. <https://doi.org/10.1145/792538.792546>