Contention Resolution with Log-Logstar Channel Accesses

Michael A. Bender
Stony Brook University, USA
bender@cs.stonybrook.edu

Tsvi Kopelowitz
University of Michigan, USA
kopelot@gmail.com

Seth Pettie
University of Michigan, USA
pettie@umich.edu

Maxwell Young
Mississippi State Univ., USA
myoung@cse.msstate.edu

ABSTRACT

For decades, randomized exponential backoff has provided a critical algorithmic building block in situations where multiple devices seek access to a shared resource. Surprisingly, despite this history, the performance of standard backoff is poor under worst-case scheduling of demands on the resource: (i) subconstant throughput can occur under plausible scenarios, and (ii) each of N devices requires $\Omega(\log N)$ access attempts before obtaining the resource.

In this paper, we address these shortcomings by offering a new backoff protocol for a shared communications channel that guarantees expected constant throughput with only $O(\log(\log^* N))$ access attempts in expectation. Central to this result are new algorithms for *approximate counting* and *leader election* with the same performance guarantees.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: [Computations on discrete structures, sequencing and scheduling]; C.2.0 [**Computer-Communication Networks**]: [Data communications]

Keywords

Distributed computing, exponential backoff; energy efficiency; multiple-access channel; randomized backoff.

1. INTRODUCTION

Randomized exponential backoff [37] is a classic algorithm for resolving contention when there is a collection of devices that all need to broadcast on a channel, but only one device can successfully broadcast at a time. If two or more devices try to broadcast on the channel simultaneously, the broadcasts interfere with each other and only noise gets transmitted. This channel might be a communication channel, such as one finds in Ethernet [37], but it could be any resource for which devices require temporary exclusive access.[1] In this paper we design a substitute for randomized exponential backoff that is globally efficient (in terms of throughput), locally efficient (in terms of per device costs), and impervious to adversarially scheduled inputs.

1.1 The Multiple-Access Channel Model

In classic algorithmic analyses of randomized backoff, there are N indistinguishable ***players*** that arrive over time, each of which needs to transmit a ***packet*** on a so-called ***multiple-access channel***.[2] Time is divided into discrete ***slots***. We assume that in each time slot we can convey a single packet of at least $\log N$-bits.[3]

We assume an ***adaptive*** adversary, which controls precisely how many players are injected into the system in each slot. In making its decisions, the adaptive adversary has access to the entire state of the system so far, the internal state of each player, but not the outcomes of future coin tosses. We do *not* assume a universal numbering scheme for the slots, that is, a global clock from which a newly injected player can infer the lifetime of the system, slot parity, etc.

In each time slot, each player in the system can do one of four actions, (i) sleep, (ii) send their packet, (iii) send a packet-sized message—a $\log N$-bit number, or (iv) listen to the channel. Players who take actions (ii), (iii), or (iv) are said to ***access*** the channel. If no players send, then the slot is ***empty***; if two or more players send, the slot is ***full*** and ***noisy***; if *exactly* one player sends, then the slot is ***full***

This research was supported by NSF grants CCF-1217708, IIS-1247726, IIS-1251137, CNS-1408695, CCF-1439084, CCF-1217338, CNS-1318294, CCF-1514383, CNS-1318294, and CCF-1420911, and Sandia National Laboratories.

STOC'16, June 19–21, 2016, Cambridge, MA, USA
© 2016 ACM. 978-1-4503-4132-5/16/06...$15.00
http://dx.doi.org/10.1145/2897518.2897655

[1]Randomized exponential backoff is implemented in a broad range of applications including local-area networks [37] wireless networks [36,54], transactional memory [33], lock acquisition [43], email retransmission [11,19], congestion control (e.g., TCP) [34,39], and a variety of cloud computing scenarios [27,41,50].

[2]Although it is usually fine to conflate the player with its associated packet, we find it useful to distinguish them. For example, in our protocols, players are obliged to do more than *merely* guarantee that their packet is sent.

[3]The notation log indicates the logarithm base 2.

and the transmission is ***successful***. All players who access the channel (actions (ii), (iii), and (iv)) can distinguish these three outcomes, and if the transmission is a successful one of type (iii), all listeners learn the $\log N$-bit number. A player is allowed to exit the system only after its packet has been successfully transmitted.

There are several axes along which we can evaluate a protocol. A global measure of effectiveness is ***utilization***, defined as the ratio of the number of slots for which at least one of the N players is in the system divided by N, and its reciprocal, which is called ***throughput***. From a single player's perspective, there is some nonzero cost to access the channel (rather than take action (i), sleeping) and depending on the technology, the cost of sending (actions (ii) and (iii)) may be different from the cost of listening (action (iv)). Our goal is to design protocols that guarantee constant expected utilization and that minimize the expected number of channel accesses per player.

Our second objective (access cost) is meaningful in the multitude of applications where the channel-access cost is high enough to be worth conserving. For example, in a wired network, an unsuccessful transmission wastes bandwidth. In a wireless network, in which devices are battery powered, sending and listening takes energy and reduces battery life. In transactional memory, a transaction rollback (i.e., an unsuccessful attempt) wastes CPU cycles.

1.2 Results

In this paper, we prove the following theorem.

Theorem 1. *There exists a randomized contention-resolution protocol enabling N online players to transmit their packets on a multiple-access channel such that:*

- *An expected constant fraction of the slots have successfully transmitted packets.*
- *Each player sends on the channel $O(1)$ times in expectation.*
- *Each player listens on the channel $O(\log(\log^* N))$ times in expectation.*

Moreover, these bounds hold even if the arrival times of the players are controlled by an adaptive adversary.

Theorem 1 contrasts with other randomized backoff protocols in the literature, which, for unknown N, require at least polylog(N) channel accesses.

One of the new ideas in our algorithm are circuits for cheaply computing functions of binary data encoded as full/empty slots. Randomness is used inside our circuit evaluation protocol for load balancing, but the evaluation itself has zero probability of error. Theorem 2 summarizes this independent component.

Theorem 2. *Let C be a Boolean circuit with ℓ bits of input and c constant-fan-in gates. There exists a protocol for n players on a multiple-access channel (n being unknown) for evaluating C, where the input is represented by ℓ specified slots (empty=0, full=1), whose sending and listening cost per player is $O(1 + \frac{c}{n})$ in expectation.*

We also give an extremely efficient protocol for estimating N and for leader-election.

Theorem 3. *There exists a randomized protocol enabling N players with a synchronized protocol-start time to (1) estimate N to within a constant factor and (2) elect a leader, such that*

- *each player sends on the channel $O(1)$ times in expectation, and*
- *each player listens on the channel $O(\log(\log^* N))$ times in expectation.*

1.3 Related Work

One variant of exponential backoff can be described as selecting slots within *windows*. When a new player is injected into the system it partitions future time into consecutive windows of length $W_0, W_1, W_2, \ldots$. If, at the beginning of window W_i, the player has yet to transmit its packet successfully, it attempts to do so at a slot in W_i chosen uniformly at random. Bender et al. [5] analyzed such backoff schemes more generally. The main take-away messages from [5] are that no *monotone* backoff strategy (in which $W_{i+1} \geq W_i$) has constant throughput, but when all the players start at once, a simple *nonmonotone* strategy called ***sawtooth*** does have constant throughput.[4] These backoff strategies only access the channel by sending, not listening, and their sending cost is $\Omega(\log N)$ in expectation.

Queuing theory arrivals. For many years, most of the analytic results on backoff assumed statistical queuing-theory models and focused on the question of what packet-arrival rates are stable (see [23,24,26,31,32,42]). Interestingly, even with Poisson arrivals, there are better protocols than binary exponential backoff, such as polynomial backoff [31].

The notion of ***saturated throughput***—roughly, the maximum throughput under stable packet arrival rates—has been examined [12, 51]. The guarantees in our paper are much stronger because we guarantee constant utilization for *arbitrarily large* arrival rates.

Worst-case/Adversarial-queueing theory arrivals. More recently, there has been work on adversarial queueing theory, looking at the worst-case performance of these protocols [1, 2, 5, 6, 15, 16, 22, 25, 28, 53]. A common theme throughout these papers, however, is that dynamic arrivals are hard to cope with. When all the players begin at the same time, very efficient protocols are possible [2,5,6,21,22,28,29,53]. When players begin at different times, the problem is much harder. The dynamic-arrival setting has been explicitly studied in the context of the wake-up problem [13, 14, 17], which looks at how long it takes for a single transmission to succeed when packets arrive dynamically.

First successful transmission/estimating ***N***. Willard [53] considered a contention-resolution problem in which the goal is to minimize the *first* moment that some player transmits successfully. Sharp time bounds of $\Theta(\log\log N)$ (in expectation) are proved when the N players begin at the same time. In [35], for any constant $\varepsilon > 0$, the authors provide an algorithm with $O((\log\log N)^{\varepsilon})$ cost for achieving a constant-factor estimation of N, with high probability. In their model, no collision detection is assumed, which makes their problem harder to deal with compared to ours.

[4]The *backoff-backon* idea behind sawtooth has been discovered in other contexts in the past [20, 30].

Adversarial fault tolerance. A number of elegant results exist on contention resolution when the channel is subject to (possibly malicious) noise [3, 40, 44–48]. A recent result by Bender et al. [7] also addresses worst-case online arrivals of players and, in the face of an unknown T noisy slots scheduled by an adaptive adversary, achieves expected constant utilization with an expected $\text{polylog}(N+T)$ number of broadcasts.

Relationship to balanced allocations. Scalable backoff is closely related to balls-and-bins games [4, 8–10, 18, 38, 49, 52]. Bins correspond to time slots and balls correspond to players. The objective is for each ball to land in its own bin; if several balls share the same bin, they are rethrown. The flow of time gets modeled by restrictions on when balls get thrown and where they may land. As mentioned earlier, listening during a slot corresponds to observing whether balls landed in a bin, and decoding an $O(\log N)$-bit number, if a single ball landed in the bin. Our results show that, remarkably, we can achieve an expected $O(\log\log^* N)$ throws and bin observations, while still achieving a constant utilization.

1.4 Algorithm Overview

At any moment in a contention-resolution protocol we can measure the ***contention*** in a single slot by summing up the probabilities of each player sending in that slot. In order to have constant throughput, we need the contention on a constant fraction of the slots to be constant. If there are currently $n \le N$ players in the system, a natural way to achieve constant contention at a single slot is to have all players send with probability $\Theta(1/n)$, which requires that players estimate n, either implicitly (via some kind of backoff) or explicitly. The first difficulty is that n is unknown and unbounded. The second is that n is constantly changing: players successfully transmit their packets and leave the system, and the adversary injects new players into the system.

We give an efficient protocol for estimating n when the n players start at the same time, that is, they agree on a slot zero. Once all n players have a mutually agreed-upon estimate $\tilde{n}$ of n, they can skip *directly* to the proper iteration of the sawtooth algorithm and finish in $\Theta(\tilde{n})$ slots with probability $1 - 1/\text{poly}(n)$, each player sending in $O(1)$ slots in expectation. The sawtooth is *truncated* on both sides: a prefix of the execution is skipped (because we know an estimate of n) and the algorithm ends after $\Theta(\tilde{n})$ slots regardless of whether a few packets still need to be transmitted.

The protocol outlined above demands perfect synchronization. Even if the players are out of step by just one time slot, the protocol could fail.[5] Thus, in our algorithms, we develop new mechanisms to achieve synchronization. Although the players arrive at various times, they organize themselves into ***batches***. In each batch, all players have an agreed-upon time zero. In each batch, the players run a protocol for estimating n and then run truncated sawtooth.

Batches never run concurrently. Consider the point of view of a new player entering the system. Because of the nonconcurrency requirement, this player must quickly distinguish between two situations: (i) no batch is currently running, in which case one can be started, possibly with other players waiting in the system, or (ii) some batch is running, in which case the player must determine precisely when it will end so that a new batch can be started immediately afterward. All the players participating in a batch will collectively guarantee that at least one slot in every constant number of consecutive slots is filled with ***humming***; humming indicates that a batch is currently running and prevents new players from starting a concurrent batch.

The protocol components are given in more detail below.

The players are synchronized. Suppose that $n \le N$ players enter the system at the same time and that they implicitly agree on time slot zero. The players begin with an ***Estimation Phase***, where the goal is to compute a $\Theta(1)$-approximation $\tilde{n}$ of n collectively.

First we address the simpler problem of testing whether or not n is close to 2^i. In a series of $\Omega(i)$ slots, each player makes noise in each slot with probability 2^{-i}. If n is close to 2^i, Chernoff bounds imply that a large constant fraction of the slots are empty *and* a large constant fraction of the slots are full. In contrast, if n is far from 2^i, then the slots are either mostly full or mostly empty. One can construct a $\Theta(i)$-sized circuit C_i that counts the number of 1 inputs (full slots=1, empty slots=0) and compares this sum against a threshold, which implies (see Theorem 2) that the players can cheaply test whether or not n is close to 2^i.

Suppose that we have deduced that $n \ge X$. We build circuit testers that test n against powers of 2 from $2^{\log X}, 2^{1+\log X}, \ldots, 2^{X^{1/3}}$. The aggregate size of these circuits is $O(X^{2/3}) = O(n^{2/3})$, so they can *all* be simulated cheaply using Theorem 2. If no circuit outputs 1 then we have a better lower bound on n, namely $n > 2^{X^{1/3}}$. Now we can afford to simulate a much larger collection of circuits in the next iteration.

On the other hand, if n is in the range $[X, 2^{X^{1/3}}]$, then some subset of the players will learn an estimate of $n \approx 2^i$, namely, the players that simulated the output gate of the circuit C_i. In order for this subset of players to notify everyone else that $n \approx 2^i$, they first elect a leader, using another $O(X^{2/3})$ slots. In a designated slot, the leader announces "2^i" and every other player listens. The total expected cost for any player to simulate a batch of circuits and elect a leader is $O(1)$ channel accesses.

A sequential search for n would involve evaluating $\log^* n$ batches of circuits. To speed up this algorithm the n players initially perform a straightforward exponential and then binary search for $\log^* n \pm 1$, which requires $O(\log(\log^* n))$ slots and channel accesses per player. Remarkably, this estimate of $\log^* n$ implies a decent enough lower bound on n, that the players need only simulate $O(1)$ batches of circuits. The Estimation Phase is described in detail in Section 4.

The process of estimating n may yield a significant underestimate or overestimate. Additionally, even if the estimate is accurate, a leader may fail to be elected. In either case, the probability of such a bad event is very small and is detectable by the players who then remedy the situation by restarting at the beginning of the Estimation Phase. These restarts do not change the players' expected cost.

The *sawtooth* contention resolution protocol of [5] is somewhat similar to the windowed version of binary exponential backoff, but is *non*monotonic. It consists of iterations indexed by $i \ge 0$, each of which consists of windows of length

[5]In practice, there are methods for robust synchronization; however, these are implementation/protocol-specific and are orthogonal to our work.

$2^i, 2^{i-1}, 2^{i-2}, \ldots$. It is fairly straightforward to show that at any iteration $i \geq \log n + O(1)$, all n packets are successfully transmitted, and are actually transmitted within the first $\log\log n + O(1)$ windows of iteration i, with high probability. Thus, once we have an estimate $\tilde{n}$, we can jump to iteration $\log \tilde{n} + O(1)$ of the sawtooth protocol. By stopping after $\log\log \tilde{n} + O(1)$ windows, the worst-case sending cost per player is $O(\log\log \tilde{n})$, the expected sending cost is $O(1)$, and all n players finish with high probability in n. This is the ***Truncated Sawtooth Phase***.

Batches and humming. Players arrive online and must be organized into synchronized ***batches***. Each batch is a set of players that execute the Estimation and Truncated Sawtooth phases, which together define one ***epoch***. Players in the current batch are called ***active***. Our algorithm guarantees (via a humming mechanism described shortly) that ***inactive*** players injected into the system in the middle of an epoch quickly learn that fact (so they do not interfere with the current batch) and eventually learn precisely when the epoch ends so that they may become active in the next batch. Thus, epochs are disjoint. If a player does not succeed in transmitting its packet during the Truncated Sawtooth phase of its epoch, that player joins the next batch.

The players in the current batch take turns humming in a constant fraction of the slots. The first purpose of humming is to never let the channel go silent for more than $O(1)$ slots so that any new inactive player can detect if it entered in the middle of an epoch.[6] The second purpose is to notify new players precisely when the current epoch will end. In the Truncated Sawtooth the players know exactly how many slots are left, say t, and will announce "t" when it is their turn to hum. Once a new player hears "t" it can sleep for t slots, wake up, and join the next batch. If, however, the current epoch is still in the Estimation Phase, no player knows when the epoch will end. In this case they encode their current estimate of $\log^* n$ using $O(\log(\log^* n))$ slots (full=1, empty=0) so that new players can sleep until Truncated Sawtooth has begun or a better estimate of $\log^* n$ is known. The listening cost for inactive players to determine when the current epoch will end is $O(\log(\log^* n))$.

Organization

This paper is organized as follows. Section 2 explains how the players can collectively simulate a circuit, proving Theorem 2. The Estimation Phase is described in Sections 3 and 4. The Truncated Sawtooth is described and analyzed in Section 5. Section 6 describes how Humming is introduced into both the Estimation and Truncated Sawtooth phases.

2. DECENTRALIZED SIMULATION OF CIRCUITS

Let C be a constant-fan-in circuit with ℓ input bits and c gates $g_1, \ldots, g_c$, listed in some topological order. There are ℓ designated slots that encode the input to the circuit (empty=0, full=1) and n players, who must collectively evaluate C. Every player knows C and the particular topological sort $g_1, \ldots, g_c$, but not n. Our simulation scheme will make use of $\ell + 2c$ time slots. The first c slots are ***control*** slots, which allow the players to take responsibility for implementing certain gates. We guarantee that at least one player is assigned to each gate. The next ℓ time slots are the input slots, each of which is either empty or full. The last c time slots are ***circuit slots*** which encode the output bit of each gate.

Control slots. The c control slots enable the players to coordinate among themselves to guarantee that (with probability 1) each gate is simulated by at least one player.

Each player P picks one uniformly random control slot τ_P and broadcasts in that slot. Let τ'_P be the first control slot following τ_P in which some player is broadcasting, or just $c+1$ if no such slot exists. Then player P will be responsible for simulating gates $g_{\tau_P}, \ldots, g_{\tau'_P - 1}$. In order for P to know exactly which gates it is responsible for, P listens to the control slots following τ_P until another player transmits or the control slots end, thereby defining τ'_P.

Some special treatment is needed for the prefix of empty control slots, since there must be at least one player that is responsible for simulating the corresponding gates. To solve this, we have *all* of the players be responsible for simulating these gates, thereby guaranteeing that every gate is simulated.

Input slots. Each input slot is either empty or full (0 or 1), and how it got to be empty or full is not the concern of this simulation. (As we will see in Sections 3 and 4, in our application it is the players themselves who determine the input. They decide, probabilistically, whether to send during an input slot.)

Circuit slots. The ith circuit slot encodes the output of gate g_i. If player P is responsible for simulating g_i then it listens during the slots corresponding to the inputs of g_i (either input slots or previous circuit slots) and sends noise on circuit slot i if and only if the output of g_i is 1. Since all the players simulating g_i will compute the same output bit, they will all either send noise or stay silent.

LEMMA 4. *The expected number of gates that a player is responsible for is $O(1 + \frac{c}{n})$.*

PROOF. Each player P is responsible for g_{τ_P} and a fake gate g_0, and the runs of unclaimed gates immediately following g_0 and g_{τ_P}. A gate g_i is unclaimed if the ith control slot is empty. Each control slot is empty with probability $(1 - \frac{1}{c})^n < \exp(-n/c)$. Moreover, conditioned on one slot being empty, the probability that the next is empty is smaller. Thus, the expected number of empty slots in the run following control slot τ_P (or control slot 0) is less than $(1 - \exp(-n/c))^{-1} - 1$, which is negligible for $n \gg c$ and always $O(\frac{c}{n})$. $\square$

PROOF OF THEOREM 2. The circuit is correctly evaluated with probability 1. The only uncertainty is the cost paid by the players. Being responsible for a gate entails listening for $O(1)$ slots (because C was assumed to have bounded fan-in) and sending in the output slot for that gate. The number of gates that a player is responsible for is $O(1 + \frac{c}{n})$ in expectation.

Note that if a player wants to learn the output of C, it must also listen to all the circuit slots corresponding to the output gates. $\square$

[6]This aspect of humming is based on the "busy signal" idea of [7].

3. A CIRCUIT FOR TESTING $\log n$

Consider the following experiment for determining whether $i \le \log n$ of $i > \log n$. In each of ℓ slots each player sends with probability $1/2^i$. If $i = \log n$ then we expect to see $\ell(1-1/2^i)^n \approx \ell/e$ empty slots. If we see fewer empty slots we conclude $i \le \log n$; if we see more empty slots we conclude $i > \log n$. We are only concerned with the accuracy of this test when $|i - \log n|$ is a sufficiently large constant (e.g., 1). It is clearly likely to give an incorrect answer when i is very close to $\log n$. We use a circuit simulation of $\mathrm{Thr}_{\ell,(1-1/e)\ell}$ (defined below) to let the players determine the outcome of this test.

LEMMA 5. *For any natural numbers ℓ and t, there exists an $\Theta(\ell)$-gate circuit $\mathrm{Thr}_{\ell,t} : \{0,1\}^\ell \to \{0,1\}$ such that, for any binary string x of length ℓ, $\mathrm{Thr}_{\ell,t}(x) = 1$ if and only if the Hamming weight (number of 1s) of x is at most t.*

PROOF. We give a circuit construction based on counting the number of 1s in x and then comparing with the target t. The construction is based upon gates with constant fan-in and fan-out.

For the counting phase, we simulate a binary tree, where the input gates at the leaves ingest the bits in x, and the $\log \ell$ outputs at the root count up to ℓ.

The construction builds upon standard one-bit adders, which take two one-bit inputs and return two outputs (a sum bit and a carry bit). The construction also uses linear-sized, constant-fan-in-fan-out, multiple-bit adders that are based upon one-bit adders (e.g., ripple-carry adders).

The root returns ℓ in binary using $\log \ell$ outputs. At the root, there is an adder, which sums the outputs from the left and right children of the root (i.e, summing the number of 1s in the left and right halves of string x). The adder has $\Theta(\log \ell)$ gates since it adds two $\log \ell$-sized number. Thus, the recurrence for the number of gates in the tree, is $S(\ell) = 2S(\ell/2) + \log \ell = \Theta(\ell)$.

Once the Hamming weight ℓ of x is represented in binary, the circuit for comparing ℓ with t takes $O(\log \ell)$ gates, and is built upon standard one-bit comparator gates (which also have constant-fan-in-fan-out). $\square$

LEMMA 6. *Define C_i to be the circuit $\mathrm{Thr}_{\ell,(1-1/e)\ell}$, where $\ell = \Omega(i)$ is a parameter, and suppose each of the n players makes noise in each input slot of C_i with probability $1/2^i$. If $i \le \log n - 1$ then the probability that C_i returns 1 is at most $\exp(-\Omega(\ln n))$. If $i \ge \log n + 1$ then the probability that C_i returns 0 is at most $\exp(-\Omega(\ell))$.*

PROOF. Suppose $\Delta = \log n - i \ge 1$. The probability of a specific input slot being empty is $(1-2^{-i})^n = (1 - 2^\Delta/n)^n = p < e^{-2^\Delta} \le e^{-2}$. Let X be the number of empty slots, so $\mathrm{E}(X) = p\ell$. By a Chernoff bound, $\Pr[X \ge \ell/e] = \exp(-\Omega(\ell))$, which is $\exp(-\Omega(\ln n))$ if $\Delta \le 2$. For $\Delta \ge 2$ we can obtain stronger bounds by direct analysis without going through Chernoff bounds. The probability of seeing at least ℓ/e empty slots is at most

$$\begin{aligned}\binom{\ell}{\ell/e} p^{\ell/e} &< \left(\frac{e\ell}{\ell/e}\right)^{\ell/e} p^{\ell/e} && \text{since } \tbinom{x}{y} < (ex/y)^y \\ &= \exp\left(\frac{2\ell}{e} + \frac{\ell \ln p}{e}\right) \\ &= \exp\left(\frac{(2-2^\Delta)\ell}{e}\right) && \ln p < -2^\Delta \\ &= \exp(-\Omega(\ln n)) && \Delta \ge 2.\end{aligned}$$

Suppose $\Delta = i - \log n \ge 1$. The probability of an input slot being empty is $(1-2^{-i})^n = (1 - 1/(n2^\Delta))^n = p > e^{-1/2^\Delta} - o(1) > 1/2$. The expected number of empty slots is $\mathrm{E}(X) = p\ell$. By a Chernoff bound, $\Pr[X \le \ell/e] = \exp(-\Omega(\ell))$. $\square$

4. AN $O(\log \log^* n)$ PROTOCOL FOR ESTIMATING n

Exponential search. For a sufficiently large constant d, define the quickly growing sequence (X_i) as:

$$X_i = \begin{cases} d & \text{if } i = 0, \\ 2^{X_{i-1}^{1/3}} & \text{if } i \ge 1. \end{cases}$$

LEMMA 7. *Let $X_{k-1} \le n < X_k$. There exists an algorithm in which n players agree on an integer $\hat{i}$ such that the cost per player is $O(\log \hat{i}) = O(\log \log^* X_{\hat{i}})$, the probability that $\hat{i} > k+1$ (an overestimate) is at most $n/X_{\hat{i}-1}$, and the probability that $\hat{i} < k-1$ (an underestimate) is at most $e^{-n/\log^3(n)}$.*

PROOF. The n players perform the following experiment to *test a value* i. In a single time slot each player sends independently with probability $1/X_i$. If the slot is full, then the result of the experiment is that $i < k$; otherwise $i \ge k$.

The algorithm uses exponential search on the index i: the players perform repeated doubling on the index i until they find the first value whose slot is empty; then they perform binary search to find the value $\hat{i}$ such that the slot for value $\hat{i}-1$ is full while the slot for value $\hat{i}$ is empty. At this point, the algorithm declares $\hat{i} = k$.

Each player listens in every slot, regardless of whether it chooses to send. Moreover, every player knows exactly when the exponential search ends, and the value of $\hat{i}$. Each player listens in $O(\log \hat{i}) = O(\log \log^* X_{\hat{i}})$ slots and sends in $O(1)$ slots in expectation.

Suppose we are testing the value i and that $k < i$. If the slot is full then we will erroneously conclude that $\hat{i} \ge i+1 > k+1$. This occurs with probability $1-(1-1/X_i)^n \le n/X_i$. Thus, if $\hat{i} > k+1$ then when testing $\hat{i}-1$ the slot was full, which happens with probability at most $n/X_{\hat{i}-1}$. On the other hand, if we are testing the value i and $i < k-1$ then the probability of seeing an empty slot, and erroneously concluding that $\hat{i} \le i$, is $\left(1 - \frac{1}{X_i}\right)^n \le e^{-n/X_i} \le e^{-n/\log^3 X_{i+1}} \le e^{-n/\log^3 n}$. $\square$

Super-circuits.

LEMMA 8. *Let $X_{k-1} \le n < X_k$. Suppose all players agree on a value i. Then there exists an algorithm in which some* *subset L of the players learn that they are in L and agree on an integer $j \in [\log X_{i-1}, \log X_i]$ (ideally, $j \approx \log n$). If $L \neq \emptyset$, then $j \in [\log n - 1, \log n + 1]$ with probability $1 - 1/\operatorname{poly}(X_i)$. The expected cost per player is $O(1 + X_{i-1}^{2/3}/n)$. If $L \neq \emptyset$, the expected size of L is $O(1 + n/X_{i-1}^{2/3})$. Moreover, if $n < X_i/2$ then the probability that $L = \emptyset$ is $1/\operatorname{poly}(X_i)$.*

PROOF. Let C'_j be the circuit C_j from Lemma 6 with an additional input bit called the "override bit". The output of C'_j is the OR of the output of C_j and the override bit. The number of input bits to C_j is ℓ, to be specified shortly.

We construct a "super-circuit" SC_i composed of subcircuits $C'_{\log X_{i-1}}, C'_{\log X_{i-1}+1}, \ldots, C'_{X_{i-1}^{1/3}}$, which collectively test whether $n \in [X_{i-1}, X_i]$. The subcircuits are chained such that the output of C'_{j-1} is the override (input) bit of C'_j; the override bit of the first subcircuit is zero. Thus, if at least one of the subcircuits passes the threshold test, then the output of the super-circuit is 1.

The goal of super-circuit SC_i is to determine whether $\log n$ lies in the interval $[\log X_{i-1}, X_{i-1}^{1/3}]$ (by searching for a $j \in [\log X_{i-1}, X_{i-1}^{1/3}]$ and declaring $\log n \approx j$). In order to control the probability of errors, we fix ℓ, the number of input bits of each of the subcircuits $\{C_j\}$ of SC_i to be exactly $\Theta(\log(X_i)) = \Theta(X_{i-1}^{1/3})$. Thus, the number of gates in SC_i is $O(X_{i-1}^{2/3})$.

Each input slot of C_j is generated by having all players make noise with probability $1/2^j$. This allows us to apply Lemma 6. Thus, the expected cost for generating input slots for each player is $O(1)$. The super-circuit is simulated using Theorem 2, and so the simulation cost per player is expected $O(1 + X_{i-1}^{2/3}/n)$.

The set L is the set of players that simulate the output gate of the first circuit C'_j that outputs 1. Notice that the players in L necessarily know that they are in L and know the value of j. If L is established, then the expected size of L is the expected number of players to simulate a single gate, which is $O(1 + n/X_{i-1}^{2/3})$.

By Lemma 6, the error probability for any subcircuit C'_j is at most $e^{-\Omega(\log X_i)}$. Thus, if $L \neq \emptyset$ and j is established then the probability that $j \leq \log n - 1$ or $j \geq \log n + 1$ is polynomially small in X_i. Moreover, if $n < X_i/2$ then the only way in which j will not be established is if the last circuit $C'_{X_{i-1}^{1/3}}$ errs, which happens with probability $e^{-\Omega(\log X_i)}$. $\square$

Leader election. The following simple bound on leader election suffices for our needs, and it serves as a black-box protocol that we can bootstrap to generate a highly efficient leader election protocol for the entire batch of players.

LEMMA 9. *Suppose there exists a subset L of the players, where each player knows whether it belongs to L, and all players know an upper bound $W > |L|$. Then there exists an algorithm that runs in $O(\log^2 W)$ time slots, after which, with probability $1 - 1/\operatorname{poly}(W)$, a single player from L establishes itself as the unique leader, and the rest of the players from L know that a leader has been established. The worst case cost of each player in L is $O(\log^2 W)$.*

PROOF. Each member of L is aware of its membership and performs the following actions (the players not in L do not access the channel). The first player to transmit *alone* is the leader. In the first slot, each player sends with probability 1. In every one of the next $d \log W$ consecutive slots, each player sends with probability 1/2, where $d > 0$ is a sufficiently large constant; in every one of the next $d \log W$ consecutive slots, each player sends with probability 1/4, and so on, each time halving the sending probability. Each player in L listens in every slot throughout the entire execution. As a result, the leader is known to all (including the leader itself).

This continues for a total of $\log W$ iterations, at which point the sending probability is $1/W$, and the number of slots executed is $O(\log^2 W)$. Since $W > |L|$, at some iteration the sending probability is close to $1/|L|$, and so the probability of not picking a leader during that iteration is polynomially small in W. $\square$

4.1 Estimating n

In this section, we give an efficient protocol enabling n players to estimate n to within a constant factor. We prove that the expected cost to estimate n is only $O(\log \log^* n)$.

THEOREM 10. *There exists an algorithm for n synchronized players to agree on an integer j such that $j < \log n - 1$ with probability $1/\operatorname{poly}(n)$ and $j > \log n + 1$ with probability $1/\operatorname{poly}(2^j)$. The protocol lasts $O(n^{2/3})$ time with probability $1 - 1/\operatorname{poly}(n)$. In expectation, each player sends $O(1)$ times and listens $O(\log \log^* n)$ times.*

PROOF. Our algorithm for estimating n up to a constant factor makes use of Lemmas 7, 8, and 9. The estimation algorithm has three types of phases.

Exponential search Phase: In the first phase, the players attempt to estimate $\log^* n$ to within an additive constant by using Lemma 7; by the end of the phase, all the players know this estimate. Remarkably, estimating $\log^* n$ seems to be the most expensive part of estimating n.

Estimating $\log n$ Phase: In the second phase, the players attempt to use the estimate of $\log^* n$ to establish a subset L of the players that: (1) know that they are in L (and every other player not in L knows that it is not in L), and (2) have the same estimate of $\log n$ which is accurate up to some additive constant. This is established by applying Lemma 8 up to three times, as follows. Let $\hat{\imath}$ be the value established by the algorithm of Lemma 7. We first simulate $SC_{\hat{\imath}-1}$, and have all of the players listen to the output of the very last gate, which is 1 iff some subcircuit C'_j outputted 1. If it is 1, some non-empty subset L of players has estimated $\log n$ and the phase ends. Otherwise, we repeat the process by simulating $SC_{\hat{\imath}}$, and if it its last gate outputs 0, by simulating $SC_{\hat{\imath}+1}$. If $SC_{\hat{\imath}+1}$ outputs 0 we declare failure; all players restart the algorithm from the exponential search phase.

Leader election Phase: If L is non-empty then all members of L agree on an estimate $j \approx \log n$. We apply Lemma 9 in an attempt to elect a leader from L. The leader sends "$n \approx 2^j$" in a predesignated time slot, in which *all* other players listen. If no leader is elected then this slot is empty, in which case a failure has occurred and all the players restart the algorithm from the exponential search phase.

Conclusion: At the end of a successful leader election phase all the players agree on an estimate of $\log n$ which is accurate up to ± 1.

Cost analysis. We now determine the expected cost of each player in the algorithm.

The cost of the exponential search phase is $O(\log\log^* X_{\hat{i}})$ where $\hat{i}$ is the output of the algorithm in Lemma 7. Let k be such that $X_{k-1} \le n < X_k$, so $k = \Theta(\log^* n)$. Since the probability of obtaining an overestimate is negligible, the expected cost of the exponential search is $O(\log\log^* n)$.

The cost of the Estimating $\log n$ phase is a bit more involved. The phase first simulates $SC_{\hat{i}-1}$ using Lemma 8, and so the expected cost of each player is $O(X_{\hat{i}-2}^{2/3}/n)$. If $\hat{i} \le k+1$ this cost is $O(1 + X_{\hat{k}-1}^{2/3}/n) \le O(1 + n^{2/3}/n) = O(1)$. In general the probability of having an erroneous $\hat{i} > k+1$ is at most $n/X_{\hat{i}-1}$, so the expected cost per player to simulate $SC_{\hat{i}-1}$ in these circumstances is at most $\sum_{\hat{i}>k+1}(n/X_{\hat{i}-1})(1 + X_{\hat{i}-2}^{2/3}/n) = O(1)$. The probability of having an erroneous $\hat{i} < k-1$ is $\exp(-n/\log^3 n)$, and the expected cost for restarting the algorithm are negligible in this case.

By Lemma 8, either $n \ge X_{\hat{i}-1}/2$ or $\Pr(SC_{\hat{i}-1}$ outputs $0) = 1/\operatorname{poly}(X_{\hat{i}-1})$. In either case, the expected cost of simulating the next super-circuit $SC_{\hat{i}}$ is $\Pr(SC_{\hat{i}-1}$ outputs $0)\cdot O(1+X_{\hat{i}-1}^{2/3}/n) = O(1+n^{-1/3}) = O(1)$. By the same reasoning, the expected cost of simulating $SC_{\hat{i}+1}$ is $\Pr($Both $SC_{\hat{i}-1}$ and $SC_{\hat{i}}$ output $0) \cdot O(1 + X_{\hat{i}}^{2/3}/n) = O(1)$.

Let C'_j be the first subcircuit to output 1 in $SC_{i^\star}$, for some $i^\star \in \{\hat{i}-1, \hat{i}, \hat{i}+1\}$. By Lemma 6, $j \in [\log n - 1, \log n + 1]$ with probability $1 - 1/\operatorname{poly}(\max\{n, X_{i^\star}\})$. The set L of players simulating the output of C'_j know that $|L| \le W = O(X_{i^\star})$ with probability $1 - 1/\operatorname{poly}(X_{i^\star})$. By Lemma 9, L correctly elects a leader in $O(\log^2 W) = O(X_{i^\star-1}^{2/3})$ slots with probability $1 - 1/\operatorname{poly}(X_{i^\star})$. The expected cost per player P to participate in leader election is $\Pr(P \in L)\cdot O(\log^2 W) = O(1/X_{i^\star-1}^{2/3}) \cdot O(X_{i^\star-1}^{2/3}) = O(1)$. □

5. THE TRUNCATED SAWTOOTH

The sawtooth protocol of Bender et al. [5] achieves constant throughput for a single batch of n players without knowing n, but at a cost of $\Omega(\log n)$ transmission attempts per player because of the exponential search for n. However, in our protocol since the n players already have an estimate $\tilde{n}$ of n, they can jump directly to the correct iteration of the sawtooth protocol and send all packets in $O(\tilde{n})$ slots with high probability and using an expected $O(1)$ channel accesses. In this section we assume $\tilde{n} \ge n$.

The truncated sawtooth protocol. We partition $O(\tilde{n})$ slots into $\log\log\tilde{n} + O(1)$ ***windows***. The 0th window has length precisely $2\tilde{n}$ and, in general, the ith window has length $2\tilde{n}/\alpha^i$ for some $\alpha > 1$.[7] The total length of all windows is less than $2\tilde{n}(\alpha-1)^{-1} = O(\tilde{n})$. At the ith window, each player that has yet to successfully transmit its packet broadcasts in a slot chosen uniformly at random from the window. If there is no collision, the player detects this and refrains from participating in the remaining windows.

Analysis. We prove that with high probability the number of active players at the beginning of window i is at most $n/f(i)$, where f is a function to be determined. Suppose the claim holds at the beginning of window i. Since at most $n/f(i)$ of the $2\tilde{n}/\alpha^i$ slots are full, the probability that a particular player fails to send its packet is at most $\frac{n/f(i)-1}{2\tilde{n}/\alpha^i} < \frac{\alpha^i}{2f(i)}$, *independent of the choices of all other players*. Call a window ***successful*** if the number of failures is at most α times its expectation, that is, at most $\frac{n\alpha^{i+1}}{2f^2(i)}$. By a Chernoff bound, $\Pr(\text{window } i \text{ is successful}) < \exp\left(-\Omega\left(\frac{(\alpha-1)^2\cdot\alpha^i n}{2f^2(i)}\right)\right)$.

Given this definition of success we set $f(i+1) = 2f^2(i)/\alpha^{i+1}$. One can prove by induction that $f(i) = 2^{2^i-1}/\alpha^{2^{i+1}-i-2}$. Note that for small α, say $\alpha = 1.1$, $f(\log\log n + O(1)) > n$ so the first $\log\log n + O(1)$ windows drastically reduce the number of active players. However, the probability of success (as defined above) becomes very low when $f^2(i)$ is close to n. Once $f(i)$ is polynomial, say at least $n^{1/5}$, the current window size is still rather large: $2\tilde{n}/\alpha^{\log\log n + O(1)} \gg n/\log n$. Once the disparity between the number of active players and window size is this large, it is easy to see that all players successfully send their packets in the next $O(1)$ windows, with probability $1 - 1/\operatorname{poly}(n)$.

LEMMA 11. *Suppose a batch of n players run the truncated sawtooth protocol and all agree on an upper bound $\tilde{n} \ge n$. The protocol takes $O(\tilde{n})$ time slots and with probability $1 - 1/\operatorname{poly}(n)$ all n players send their packets. The sending cost per player is $O(1)$ in expectation and $\log\log\tilde{n} + O(1)$ in the worst case.*

6. HUMMING

Players arrive at arbitrary times and must be organized into batches that are ***synchronized***, that is, they all agree on a time 0. Once a batch of n players is formed, the players obtain an upper bound $\tilde{n} \ge n$ and then run the truncated sawtooth protocol in $O(\tilde{n})$ slots.

Consider the point of view of a new player entering the system. If there is no active batch running, then the player should be able to detect this and start a new batch, possibly with other players who entered at the same time. Otherwise, if there is an active batch the new player should determine *precisely* how many slots it has left, say t, so it can join a new batch beginning in $t+1$ slots. One difficulty is that players within the active batch only know how many slots are left once they compute $\tilde{n}$ and enter the sawtooth. Thus, we need a mechanism with the following properties:

- New players can determine, by listening to $O(\log\log^* n)$ slots, whether there is an active batch and, if so, precisely how many slots remain.
- The sending cost per player in the current batch should be $O(1)$ in expectation.

To achieve these ends we have to ensure that an active batch never lets the channel go silent for more than a constant number of slots at a time, because this would confuse a new player into thinking no batch is active. Thus, the n players in the current batch collectively *hum* in at least one in every constant number of slots to keep new players from joining prematurely.

Humming in the estimation phase. We imagine a ***baton*** being passed between groups of players; at time 0 in the Estimation Phase all players hold the baton. The baton holders

[7]The sawtooth algorithm of [5] fixes the decay factor α to be 2. We find it helpful in the analysis to keep α a named parameter.

are responsible for both humming *and* encoding the current estimate ρ of $\log^* n$. Until the Estimation Phase finishes its binary search to find an i for which $n \in [X_{i-1}, X_{i+1}]$, ρ is 0. When the Estimation Phase is simulating circuits that test for $n \in [X_{i-1}, X_i)$, $\rho = i$.

The time slots are partitioned into ***chunks*** of 7 consecutive slots, only one of which is dedicated to executing the Estimation Phase proper. The remaining slots implement baton passing, humming, and encoding ρ.

(0) **baton:** Any player wishing to take the baton broadcasts noise in this slot. All other current baton holders listen in this slot to determine if they still hold the baton.
(1,2) **humming:** Any player holding the baton hums for two consecutive slots.
(3) **silence:** This slot is always empty.
(4) **rho:** Whoever holds the baton sends the next bit in the encoding of ρ: noise=1 and silence=0.
(5) **silence:** This slot is always empty.
(6) **data:** This slot is used to implement the Estimation Phase algorithm.

Observe that there are never six consecutive silent slots, and that any new player can deduce the current time slot modulo 7 by listening to $O(1)$ consecutive slots: it listens for two, three, or four consecutive noisy slots (which can only be slots (1,2) or (0,1,2) or (6,0,1,2)) followed by a silent slot, which must be slot (3).

If the binary representation of ρ is $b_1 b_2 \cdots b_\ell$, the baton holders repeatedly encode ρ in the **rho** slots as $110b_1 0b_2 0 \cdots b_\ell 0$. A listener can decode ρ by listening to $4\ell + O(1) = O(\log \rho)$ chunks: at most $2\ell + O(1)$ chunks to hear two consecutive 1s and $2\ell + O(1)$ more to determine ρ.

At all times the algorithm that estimates n has computed a *likely lower bound* $\bar{n}$ on n.[8] Each player (even one that holds the baton) makes noise in the **baton** slot with probability $1/\bar{n}$, thereby taking the baton from whatever group that holds it. Thus, any player that currently holds the baton is relieved of duty with probability $(1 - 1/\bar{n})(1 - (1 - 1/\bar{n})^{n-1})$, which is $\Theta(1)$ for $n \geq \bar{n} > 1$. In the first chunk, all players hold the baton. Since the Estimation Phase lasts for $O(n^{2/3})$ slots in expectation, each player holds the baton for $O(1)$ chunks in expectation.

Humming in the truncated sawtooth.

In this phase all nodes in the active batch have agreed on an estimate $\tilde{n}$, which is $\Theta(n)$ w.h.p., and know precisely when the Truncated Sawtooth phase ends. They need to communicate this information to new players joining the system. Slots are partitioned into chunks of three. The players maintain the invariant that *exactly* one player holds the baton at any given time. The initial baton-holder is the elected leader who announced "$\tilde{n}$" at the end of the Estimation Phase.

(0) **baton:** Every non-baton holder that wishes to take the baton broadcasts a message in this slot. If *exactly* one player broadcasts, he takes the baton; if zero or more than one players broadcast, the current baton-holder retains the baton.
(1) **humming:** The (unique) baton-holder sends "t" if there are t slots remaining in the truncated sawtooth phase.
(2) **data:** This slot is used to implement the Truncated Sawtooth algorithm.

[8]In the binary search for $\log^* n \pm 1$ this lower bound can be updated in every time slot. When simulating the circuits that test for $n \in [X_{i-1}, X_i)$ we use X_{i-1} as a lower bound $\bar{n}$ on n.

Each non-baton holder attempts to take the baton in slot (0) with probability $1/\tilde{n}$. Thus, the current baton holder is relieved of duty with probability $(n-1)(1/\tilde{n})(1 - 1/\tilde{n})^{n-2}$, which is $\Theta(1)$ for $n > 1$ and $n = O(\tilde{n})$. Since there are $O(\tilde{n})$ chunks each player attempts to take the baton $O(1)$ times in expectation. By symmetry each player holds the baton for $O(\tilde{n}/n)$ chunks in expectation.

Consider how the humming protocol allows new players to cheaply determine when the current batch will finish. If a player joins during the Truncated Sawtooth, it listens for at most three slots and learns t: the exact number of remaining slots. If a player joins during the Estimation Phase it listens until it decodes an estimate $\rho > 0$, indicating that the algorithm is simulating circuits testing for $n \in [X_{\rho-1}, X_\rho)$. Let $L_\rho = O(\log^2 X_\rho)$ be the length of the simulation. In general, whenever the player decodes ρ it sleeps for L_ρ slots and begins listening again. If the Estimation Phase is successful then the new player may listen for $O(\log \rho) = O(\log \log^* n)$ slots in four intervals: it may decode a ρ, sleep for L_ρ steps, decode $\rho+1$, sleep for $L_{\rho+1}$ steps, decode $\rho+2$, sleep for $L_{\rho+2}$ steps, then listen in the Truncated Sawtooth and determine exactly how many slots remain.

Pulling the pieces together, we can now give a proof of Theorem 1:

Proof of Theorem 1. Define n_i to be the number of players that participate in the ith batch. By Theorem 10 and Lemma 11, with high probability in n_i, the Estimation and Truncated Sawtooth Phases take $O(n_i^{2/3} + n_i) = O(n_i)$ slots and transmit all packets. Moreover, each player in the batch listens in $O(\log \log^* n_i)$ slots and sends in $O(1)$ slots, in expectation. Under these circumstances the throughput is constant.

The throughput can suffer if n_i is underestimated (few packets will be sent in this epoch due to high contention) or overestimated (the epoch will likely be successful, but take too long) or if n_i is correctly estimated but collisions prevent a large number of packets from being transmitted. By Theorem 10, the probability that n_i is underestimated is $1/\operatorname{poly}(n_i)$ and the probability that it is overestimated as $\tilde{n}_i > 2n_i$ is $1/\operatorname{poly}(\tilde{n}_i)$. By Lemma 11, when n_i is correctly estimated, the probability that not all packets are sent is $1/\operatorname{poly}(n_i)$. Call a slot ***failed*** if it is in a batch that failed due to one of these three causes. Since there are $O(\tilde{n}_i)$ slots in the epoch, the expected number of failed slots per epoch is $1/\operatorname{poly}(\tilde{n}_i)$, implying that the expected throughput is constant.

A player injected during batch i will listen to $O(\log \log^* n_i)$ slots in expectation before joining batch $i+1$, then listen for $O(\log \log^* n_{i+1})$ slots in batch $i+1$, in expectation. The sending cost per player is $O(1)$ in expectation in both the Estimation and Truncated Sawtooth phases. $\square$

7. REFERENCES

[1] L. Anantharamu, B. S. Chlebus, and M. A. Rokicki. Adversarial Multiple Access Channel with Individual Injection Rates. In *Proceedings of the 13th*

International Conference on Principles of Distributed Systems (OPODIS), pages 174–188, 2009.

[2] A. F. Anta, M. A. Mosteiro, and J. R. Muñoz. Unbounded contention resolution in multiple-access channels. *Algorithmica*, 67(3):295–314, 2013.

[3] B. Awerbuch, A. Richa, and C. Scheideler. A Jamming-Resistant MAC Protocol for Single-Hop Wireless Networks. In *Proceedings of the 27th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 45–54, 2008.

[4] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, Sept. 1999.

[5] M. A. Bender, M. Farach-Colton, S. He, B. C. Kuszmaul, and C. E. Leiserson. Adversarial contention resolution for simple channels. In *Proc. 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 325–332, 2005.

[6] M. A. Bender, J. T. Fineman, and S. Gilbert. Contention resolution with heterogeneous job sizes. In *Proc. 14th Annual European Symposium on Algorithms (ESA)*, pages 112–123, 2006.

[7] M. A. Bender, J. T. Fineman, S. Gilbert, and M. Young. How to scale exponential backoff: Constant throughput, polylog access attempts, and robustness. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016.

[8] P. Berenbrink, A. Czumaj, M. Englert, T. Friedetzky, and L. Nagel. Multiple-choice balanced allocation in (almost) parallel. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX-RANDOM)*, pages 411–422, 2012.

[9] P. Berenbrink, A. Czumaj, A. Steger, and B. Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006.

[10] P. Berenbrink, K. Khodamoradi, T. Sauerwald, and A. Stauffer. Balls-into-bins with nearly optimal load distribution. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 326–335, 2013.

[11] D. J. Bernstein. qmail — an email message transfer agent. http://cr.yp.to/qmail.html, June 1998.

[12] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535–547, Sept. 2006.

[13] B. S. Chlebus, L. Gasieniec, D. R. Kowalski, and T. Radzik. On the Wake-up Problem in Radio Networks. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, pages 347–359, 2005.

[14] B. S. Chlebus and D. R. Kowalski. A Better Wake-up in Radio Networks. In *Proceedings of 23rd ACM Symposium on Principles of Distributed Computing (PODC)*, pages 266–274, 2004.

[15] B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki. Adversarial queuing on the multiple-access channel. In *Proc. Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC*, pages 92–101, 2006.

[16] B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki. Adversarial queuing on the multiple access channel. *ACM Transactions on Algorithms*, 8(1):5, 2012.

[17] M. Chrobak, L. Gasieniec, and D. R. Kowalski. The Wake-up Problem in Multihop Radio Networks. *SIAM Journal on Computing*, 36(5):1453–1471, 2007.

[18] R. Cole, A. M. Frieze, B. M. Maggs, M. Mitzenmacher, A. W. Richa, R. K. Sitaraman, and E. Upfal. On balls and bins with deletions. In *Proc. Second International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 145–158, 1998.

[19] B. Costales and E. Allman. *Sendmail*. O'Reilly, third edition, Dec. 2002.

[20] M. Geréb-Graus and T. Tsantilas. Efficient optical communication in parallel computers. In *Proceedings 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 41–48, 1992.

[21] M. Geréb-Graus and T. Tsantilas. Efficient optical communication in parallel computers. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 41–48, 1992.

[22] L. A. Goldberg, M. Jerrum, T. Leighton, and S. Rao. Doubly logarithmic communication algorithms for optical-communication parallel computers. *SIAM J. Comput.*, 26(4):1100–1119, Aug. 1997.

[23] L. A. Goldberg and P. D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. In *Proc. Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 554–563, 1996.

[24] L. A. Goldberg, P. D. Mackenzie, M. Paterson, and A. Srinivasan. Contention Resolution with Constant Expected Delay. *Journal of the ACM*, 47(6):1048–1096, 2000.

[25] L. A. Goldberg, Y. Matias, and S. Rao. An optical simulation of shared memory. *SIAM J. Comput.*, 28(5):1829–1847, Oct. 1999.

[26] J. Goodman, A. G. Greenberg, N. Madras, and P. March. Stability of binary exponential backoff. *J. ACM*, 35(3):579–602, July 1988.

[27] Google. GCM (Google Cloud Messaging) Advanced Topics. http://developer.android.com/google/gcm/adv.html#retry, 2014.

[28] A. G. Greenberg, P. Flajolet, and R. E. Ladner. Estimating the multiplicities of conflicts to speed their resolution in multiple access channels. *JACM*, 34(2):289–325, Apr. 1987.

[29] A. G. Greenberg and S. Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *JACM*, 32(3):589–596, July 1985.

[30] R. I. Greenberg and C. E. Leiserson. Randomized routing on fat-trees. In *Proceedings 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 241–249, 1985.

[31] J. Hastad, T. Leighton, and B. Rogoff. Analysis of backoff protocols for multiple access channels. In *STOC'87*, pages 241–253, New York, New York, May 1987.

[32] J. Hastad, T. Leighton, and B. Rogoff. Analysis of

Backoff Protocols for Mulitiple Access Channels. *SIAM Journal on Computing*, 25(4):1996, 740-774.

[33] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. In *Proc. of the 20th Intnl. Conference on Computer Architecture.*, pages 289–300, San Diego, California, 1993.

[34] V. Jacobson and M. J. Karels. Congestion Avoidance and Control. In *Symposium Proceedings on Communications Architectures and Protocols (SIGCOMM)*, pages 314–329, 1988.

[35] T. Jurdziński, M. Kutyłowski, and J. Zatopiański. Energy-Efficient Size Approximation of Radio Networks with No Collision Detection. In *Proceedings of the 8th Annual International Conference (COCOON)*, pages 279–289, 2002.

[36] J. F. Kurose and K. Ross. *Computer Networking: A Top-Down Approach Featuring the Internet.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[37] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed packet switching for local computer networks. *CACM*, 19(7):395–404, July 1976.

[38] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, Oct 2001.

[39] A. Mondal and A. Kuzmanovic. Removing Exponential Backoff from TCP. *SIGCOMM Comput. Commun. Rev.*, 38(5):17–28, Sept. 2008.

[40] A. Ogierman, A. Richa, C. Scheideler, S. Schmid, and J. Zhang. Competitive MAC under Adversarial SINR, 2013. http://arxiv.org/abs/1307.7231.

[41] G. A. Platform. Google Documents List API version 3.0: Implementing Exponential Backoff. https://developers.google.com/google-apps/documents-list/?csw=1#implementing_exponential_backoff, 2011.

[42] P. Raghavan and E. Upfal. Stochastic contention resolution with short delays. *SIAM J. Comput.*, 28(2):709–719, Apr. 1999.

[43] R. Rajwar and J. R. Goodman. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *Proc. of the 34th Annual Intnl. Symposium on Microarchitecture*, pages 294–305, Austin, Texas, Dec. 2001.

[44] A. Richa, C. Scheideler, S. Schmid, and J. Zhang. A Jamming-Resistant MAC Protocol for Multi-Hop Wireless Networks. In *Proceedings of the International Symposium on Distributed Computing (DISC)*, pages 179–193, 2010.

[45] A. Richa, C. Scheideler, S. Schmid, and J. Zhang. Competitive and Fair Medium Access Despite Reactive Jamming. In *Proceedings of the* 31^{st} *International Conference on Distributed Computing Systems (ICDCS)*, pages 507–516, 2011.

[46] A. Richa, C. Scheideler, S. Schmid, and J. Zhang. Competitive and Fair Throughput for Co-Existing Networks Under Adversarial Interference. In *Proceedings of the* 31^{st} *ACM Symposium on Principles of Distributed Computing (PODC)*, 2012.

[47] A. Richa, C. Scheideler, S. Schmid, and J. Zhang. An Efficient and Fair MAC Protocol Robust to Reactive Interference. *IEEE/ACM Transactions on Networking*, 21(1):760–771, 2013.

[48] A. Richa, C. Scheideler, S. Schmid, and J. Zhang. Competitive Throughput in Multi-Hop Wireless Networks Despite Adaptive Jamming. *Distributed Computing*, 26(3):159–171, 2013.

[49] A. W. Richa, M. Mitzenmacher, and R. Sitaraman. The power of two random choices: A survey of techniques and results. *Combinatorial Optimization*, 9:255–304, 2001.

[50] A. W. Services. Error Retries and Exponential Backoff in AWS. http://docs.aws.amazon.com/general/latest/gr/api-retries.html, 2012.

[51] N.-O. Song, B.-J. Kwak, and L. E. Miller. On the stability of exponential backoff. *Journal of Research of the National Institute of Standards and Technology*, 108(4), 2003.

[52] B. Vöcking. How asymmetry helps load balancing. *J. ACM*, 50(4):568–589, 2003.

[53] D. E. Willard. Log-logarithmic Selection Resolution Protocols in a Multiple Access Channel. *SIAM J. Comput.*, 15(2):468–477, May 1986.

[54] Y. Xiao. Performance Analysis of Priority Schemes for IEEE 802.11 and IEEE 802.11e Wireless LANs. *Wireless Communications, IEEE Transactions on*, 4(4):1506–1515, July 2005.