How to Scale Exponential Backoff: Constant Throughput, Polylog Access Attempts, and Robustness*

Michael A. Bender† Jeremy T. Fineman‡ Seth Gilbert§ Maxwell Young¶

Abstract

Randomized exponential backoff is a widely deployed technique for coordinating access to a shared resource. A good backoff protocol should, arguably, satisfy three natural properties: (i) it should provide constant throughput, wasting as little time as possible; (ii) it should require few failed access attempts, minimizing the amount of wasted effort; and (iii) it should be robust, continuing to work efficiently even if some of the access attempts fail for spurious reasons. Unfortunately, exponential backoff has some well-known limitations in two of these areas: it provides poor (sub-constant) throughput (in the worst case), and is not robust (to adversarial disruption).

The goal of this paper is to "fix" exponential backoff by making it scalable, particularly focusing on the case where processes arrive in an on-line, worst-case fashion. We present a relatively simple backoff protocol, Re-Backoff, that has, at its heart, a version of exponential backoff. It guarantees expected constant throughput with dynamic process arrivals and requires only an expected polylogarithmic number of access attempts per process.

Re-Backoff is also robust to periods where the shared resource is unavailable for a period of time. If it is unavailable for D time slots, Re-Backoff provides the following guarantees. When the number of packets is a finite n, the average expected number of access attempts for successfully sending a packet is $O(\log^2(n+D))$. In the infinite case, the average expected number of access attempts for successfully sending a packet is $O(\log^2(\eta+D))$ where η is the maximum number of processes that are ever in the system concurrently.

*Supported in part by NSF grants CCF 1114809, CCF 1217708, CCF 1218188, CCF 1314633, IIS 1247726, IIS 1251137, CNS 1408695, CCF 1439084, CNS-1318294, CCF-1420911, by Sandia National Laboratories, and by Singapore NUS FRC-T1-251RES1404.

†Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-2424, USA. Email: bender@cs.stonybrook.edu.

‡Department of Computer Science, Georgetown University, USA. Email jfineman@cs.georgetown.edu.

§Department of Computer Science, National University of Singapore, Singapore. Email: seth.gilbert@comp.nus.edu.sg.

¶Department of Computer Science and Engineering, Mississippi State University, USA. Email: myoung@cse.msstate.edu.

1 Introduction

Randomized exponential backoff [42] is used throughout computer science to coordinate access to a shared resource. This mechanism applies when there are multiple processes (or devices, players, transactions, or packets) attempting to access a single, shared resource, but only one process can hold the resource at a time. Randomized backoff is implemented in a broad range of applications including local-area networks [42] wireless networks [40, 63], transactional memory [36], lock acquisition [50], email retransmission [10, 19], congestion control (e.g., TCP) [38, 45], and a variety of cloud computing scenarios [30, 47, 57].

In randomized exponential backoff, when a process needs the resource, it repeatedly attempts to grab it. If two processes ***collide***—i.e., they try to grab the resource at the same time—then the access fails, and each process waits for a randomly chosen amount of time before retrying. After each collision, a process's expected waiting time doubles, resulting in reduced contention and a greater probability of success.

When a process needs the resource:
- Set the time window size $W = 2$.
- Repeat until the resource is acquired:
  - Choose a slot t in window W uniformly at random. Try to acquire the resource at slot t.
  - If the acquisition failed, then: (i) wait until the end of W, and (ii) set $W = 2W$.

Figure 1: Exponential backoff.

Given the prevalence (and elegance) of exponential backoff, it should not be surprising that myriad papers have studied the theoretical performance of randomized backoff. Many of these papers make queuing-theory assumptions on the arrival of processes needing the resource [12, 22, 26, 34, 48]. Others assume that all processes arrive in a batch [21, 23, 31, 32, 62] or adversarially [5, 15, 16]. What may be surprising is how many foundational questions about randomized backoff remain unanswered, or only partially answered:

- *Throughput:* It is well known that classical exponential backoff achieves sub-constant throughput, in the

worst case. Is it possible for an exponential backoff variant to achieve constant throughput, particularly in dynamic settings, where arbitrarily large bursts of processes may arrive in any time step, leading to varying resource contention over time?

- *Number of attempts:* On average, how many attempts does a process make before it successfully acquires the resource? Can one modify exponential backoff to achieve constant throughput, while still ensuring a few unsuccessful attempts? These questions make sense in applications where each access attempt has a cost. For example, in a wired network, an unsuccessful transmission wastes bandwidth. In a wireless network, an unsuccessful transmission wastes energy. In transactional memory, a transaction rollback (i.e., an unsuccessful attempt) wastes CPU cycles.

- *Robustness:* How robust is exponential backoff when the acquisition process suffers failures? An access attempt could fail *even when there is no collision.* Faults could arise due to hardware failures, software bugs, or malicious behavior. For example, a shared link may suffer from thermal noise, or a heavily-loaded server may crash. A wireless channel may come under attack from jamming (see [61, 64]), or a server may become unavailable due to a denial-of-service attack (DoS) [37]. In transactional memory, failures may result from best-effort hardware, since existing hardware implementations do not guarantee success even when there are no collisions (conflicts) between transactions.

1.1 Goal: Make Exponential Backoff Scalable

Randomized exponential backoff is "broken" in the worst case: it lacks good throughput guarantees (see, e.g., [5]) and is not robust to failures. While randomized exponential backoff permits a relatively small number of access attempts, the result is a subconstant throughput.

The goal of this paper is to "fix" randomized exponential backoff to achieve good *asymptotic* performance. We modify the protocol as little as possible to maintain its simplicity, while finding a variant that (1) delivers constant throughput, (2) requires few access attempts, and (3) works robustly.

Since exponential backoff has many applications, there are many choices of terminology. Here we call the shared resource the "channel" and attempts to acquire the resource "broadcasts."[1]

[1]We permit some slight abuses of the English language when packets may seem to broadcast themselves.

1.2 Related Work

Exponential backoff is commonly studied in a network setting, where packets arriving over time are transmitted on a ***multiple-access channel***.

Queuing models. For many years, most backoff analyses assumed statistical-queueing-theory models and focused on finding stable packet-arrival rates (see [1,25,27,29,34,35,49]). In this context, the notion of *saturated throughput* — roughly, the maximum throughput under stable packet arrival rates — has been investigated [11,59]. In contrast, one might desire a stronger guarantee on the worst-case throughput regardless of the arrival rate (which is what we address here). Interestingly, even with Poisson arrivals, there are better protocols than binary exponential backoff, e.g., polynomial backoff [34]. The world runs on exponential backoff; nonetheless, it has long been known that exponential backoff is symptotically sub-optimal.

Worst-case/adversarial arrivals. More recently, there has been work on adversarial queueing theory, looking at the worst-case performance [2, 5, 6, 15, 16, 20, 24, 28, 31, 62]. A common theme is that dynamic arrivals are hard to cope with. When all the packets begin at the same time, efficient protocols are possible [20, 21, 23, 31, 32, 62]. When packets begin at different times, the problem is harder. Dynamic arrivals 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. In contrast, our paper focuses on achieving good bounds for a stream of packet arrivals (no fixed stations), when all packets must be transmitted.

Robustness to wireless interference. As the focus on backoff protocols has shifted to include wireless networks, there has been an increasing emphasis on coping with noise and interference known as *jamming.* In a surprising breakthrough, Awerbuch et al. [3] showed that good throughput is possible with a small number of access attempts, even if jamming causes disruption for a constant fraction of the execution. A number of elegant results have followed [39, 46, 51–55], with good guarantees on throughput and access attempts.

Most of these jamming-resistant protocols do not assume fully dynamic packet arrivals. By this, we mean that these protocols are designed for the setting in which there are a fixed number of "stations" that are continually transmitting packets. In contrast, we are interested in the fully dynamic setting, where sometimes there are arbitrarily large bursts of packets arriving (lots of channel contention) and other times there are lulls with small handfuls of packets (little channel contention).

Relationship to balanced allocations. Scalable backoff is closely related to balls-and-bins games [4, 7–9, 18, 43, 56, 60]. Bins correspond to time slots and balls correspond to packets. 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. The results in our paper are ultimately about scalable randomized algorithms and asymptotic analysis for dealing with bursts robustly and scalably.

1.3 Results

We devise a "(R)obust (E)fficient" backoff protocol, Re-Backoff, that (1) delivers constant throughput, (2) guarantees few failed access attempts, and (3) works robustly. We assume no global broadcast schedule, shared secrets, or centralized control.

Theorem 1.1. *Let D be the number of slots disrupted by the adversary. For a finite number of packets n injected into the system, where n is fixed* a priori*, but not revealed,* Re-Backoff *guarantees at most an expected constant fraction of wasted slots (empty slots or slots with collisions) and spends $O(\log^2(n + D))$ access attempts per packet, in expectation.*

Theorem 1.1 implies that Re-Backoff delivers constant throughput for those executions where at least a constant fraction of the slots are undisrupted:

Corollary 1.1. *There exists a constant c, such that if $D \leq cn$, then* Re-Backoff *achieves expected constant throughput. A stronger property holds:* Re-Backoff *attains an expected makespan (completion time of last packet) of $O(n)$.*

In fact, the metric of expected makespan is both stronger and more desirable than expected throughput. To see why, observe that throughput is defined as the fraction of successful slots; see Section 2.2. Consider a scenario in which the throughput is $1/2$ with probability $1/2$ and 0 with probability $1/2$: then the expected throughput is constant, but the expected makespan is infinite. Really, we want the the expected *reciprocal* of throughput to be $O(1)$, which is equivalent to saying that the expected makespan is $O(n)$. In this paper, we focus on minimizing the expected makespan, optimizing the expected throughput as a byproduct.

An implication of Theorem 1.1 is that the number of access attempts is small relative to the number of disrupted slots. Specifically: (1) Our protocol is parsimonious with broadcast attempts in the absence of disruption. (2) If a packet has been in the system for T slots, then it makes expected $O(\log^2 T)$ access attempts, regardless of how many of these T slots were adversarially blocked.

Extending these results to the infinite case, we show that in an infinite execution, for a countably infinite number of slots, the protocol achieves both good throughput and few access attempts.

Theorem 1.2. *For any time t, denote by D_t the number of slots disrupted before t, and by η_t the maximum number of packets concurrently in the system before t. Suppose an infinite number of packets get injected into the system. Then for any time s, there exists a time $t \geq s$ such that at time t,* Re-Backoff *has at most constant waste with probability 1 and expected average $O(\log^2(\eta_t + D_t))$ access attempts per packet.*

Again, this implies that Re-Backoff yields constant throughput in infinite executions:

Corollary 1.2. *There exists a constant c such that: for every time s there exists a time $t \geq s$ where if $D_t \leq ct$, then there is constant throughput until time t.*

2 Model: Contention Resolution on a Multiple-Access Channel

Time is discretized into ***slots*** where a process can broadcast a packet, i.e., access the shared resource. We do not assume a global clock, i.e., there is no universal numbering scheme for slots.

When there is no transmission (or adversarial disruption, see below) during a slot, we call that slot ***empty***. A slot is ***full*** when one or more packets are broadcast in that slot. When exactly one packet is broadcast in a slot, that packet transmits successfully, and the full slot is ***successful***. When two or more packets are broadcast during the same slot, a ***collision*** occurs in this (full) slot. When there is a collision, there is "noise" on the channel; all packets transmitting are unsuccessful.

We assume that a device transmitting a packet can determine whether its transmission is successful; this is a standard assumption in the backoff literature (for examples, see [27, 35, 41]), unlike the wireless setting where a full medium access control (MAC) protocol would address acknowledgments and other issues. Here, (as in exponential backoff), we focus solely on the sending side (backoff component) of the problem.

We also assume that a device that is listening on the channel, but not transmitting, can tell whether a given slot is full or empty. We do not require the listener to distinguish between collisions and successful transmissions.

For simplicity of presentation, we assume there are actually two channels that processes can use simultaneously: a "control channel" and a "data channel." We

explain in Section 7 how to implement our solution using only one channel.

2.1 Arbitrary Dynamic Packet Arrivals

New packets arrive arbitrarily over time. We do not assume any bounded arrival rate. A packet is ***live*** at any time between its arrival and its successful transmission. The number of packets in the system may vary arbitrarily over time, and this number is unknown to the packets. Without loss of generality, we assume that throughout the execution of the protocol, there is at least one live packet in every slot. (If not, simply ignore any slot during which there are no live packets.)

We postulate an ***adversary***, who governs two aspects of the system's dynamics. (1) The adversary determines the (finite) number of new packets that arrive in each slot. (2) The adversary may arbitrarily ***disrupt*** slots. A disruption appears as a full slot to all packets; any packet transmitted simultaneously fails. This model corresponds to collisions on Ethernet or a 1-uniform adversary in wireless networks (see [51]).

The adversary is adaptive with one exception—the adversary decides *a priori* whether the execution contains infinitely many packets or a finite number n of packets. For the finite case, the adversary chooses n *a priori*. The packets themselves do not know whether the instance is infinite or finite (meaning that they cannot know n). In all other ways, the adversary is adaptive: it may make all arrival and disruption decisions with full knowledge of the current and past system state; at the end of a given slot, the adversary learns everything that has happened in that slot.

2.2 Throughput and Waste

We define throughput in the natural way: for an interval I, the throughput $\lambda \in [0,1]$ is the fraction of successful slots in the interval I. (Recall that for the purposes of throughput and waste, we only consider slots when there is at least one packet live in the system.)

We also define a notion of "waste." A slot is ***wasted*** if there was a missed opportunity for a successful transmission: the slot was empty or more than one packet was broadcast. Otherwise the slot is ***nonwasted***, i.e., successful or disrupted. A disrupted slot is not seen as wasted, since it could never be used for a successful packet transmission. The ***nonwaste*** $\Lambda \in [0,1]$ of an interval I is the fraction of nonwasted slots in I, and the ***waste*** is $1-\Lambda$. In the absence of disruption, throughput and nonwaste are identical.

Definition 1. *Consider interval $\mathcal{I}$ having $N_{\mathcal{I}}$ successful transmissions and $D_{\mathcal{I}}$ disrupted slots. The* ***throughput*** *of $\mathcal{I}$ is $\lambda = N_{\mathcal{I}}/|\mathcal{I}|$, the* ***nonwaste*** *is $\Lambda = (N_{\mathcal{I}} + D_{\mathcal{I}})/|\mathcal{I}|$, and the* ***waste*** *is $1-\Lambda$. The throughput/waste of a finite execution is the throughput/waste for the interval $[0,T]$, where T is the latest any packet completes.*

Definition 2. *An infinite instance has Λ-nonwaste if, for any slot t, there exists a slot $t' \geq t$ where interval $[0,t']$ has Λ nonwaste. An infinite instance has λ-throughput if, for any slot t, there exists a slot $t' \geq t$ where interval $[0,t']$ has λ throughput.*

The throughput/nonwaste does not depend on the arrival rate, even with no restrictions on arrivals. The arrival rate could be higher than feasible for an arbitrary period of time (e.g., two packets arrive every slot), and the system continues to deliver good throughput (even as the number of backlogged packets necessarily grows).

There are also no restrictions on the distribution of disruptions. The adversary can choose to disrupt arbitrarily large intervals of slots. When there are enough nondisrupted slots, constant throughput resumes.

3 Algorithm

This section presents our backoff protocol. To simplify the presentation, we assume throughout that there are two communication channels, a ***data channel***, on which packets are broadcast, and a ***control channel***, on which a "busy signal" is broadcast. See Section 7 for how to implement this algorithm on one channel.

For a given packet u, let s_u be the number of slots it has been active for. Our protocol has the following structure (see Figure 2 for pseudocode):

- Initially, each packet is ***inactive***; it makes no attempt to broadcast on either channel.
- Inactive packets monitor the control channel. As soon as the packet observes an empty slot on the control channel, it becomes ***active***.
- In every time slot, an active packet broadcasts on the data channel with probability proportional to how long it has been active, i.e., packet u broadcasts with probability d/s_u, for a constant $d = 1/2$. It also broadcasts on the control channel with probability $c \max(\ln s_u, 1)/s_u$, for a constant $c > 0$.
- A packet remains active until it transmits successfully or sees too many empty slots. Specifically, if packet u has observed $\lceil \gamma s_u \rceil$ empty slots, where $\gamma = 15/16$, then the packet reverts to an ***inactive*** state, and the process repeats.

In essence, our protocol wraps exponential backoff with a coordination mechanism (i.e., the busy channel) to limit entry, and with an abort mechanism to prevent overshooting. In between, it runs something akin

RE-BACKOFF for a packet u that has been active for s_u slots

- With probability $\frac{c \max\{\ln s_u,\ 1\}}{s_u}$, send busy tone on the control channel.
- With probability $\frac{d}{s_u}$ send m_u on the data channel and, if successful, then terminate.
- Monitor the data channel. If at least $\lceil \gamma s_u \rceil$ data slots have been empty ($\gamma = 15/16$) since node u became active, then become inactive.

RE-BACKOFF for an inactive packet

- Monitor each control slot. If a slot is empty, then become active in the next slot.

Figure 2: Pseudocode for RE-BACKOFF.

to classical exponential backoff (instantiated by broadcasting in round t with probability $1/t$, instead of using windows). One aspect that we find interesting is how little it takes to fix exponential backoff.

4 Protocol Design

This section gives the intuition behind the design of RE-BACKOFF.

Consider the following simple protocol that RE-BACKOFF builds upon. Packets are initially inactive. Whenever there are no active packets, all packets in the system become active and run an asymptotically optimal ***batch*** backoff protocol on the data channel (e.g., SawTooth Backoff [5]). Active packets *all* broadcast a ***busy tone*** on every control-channel slot, and inactive packets wait for silence on the control channel.[2] The busy tone contains *no data*, and it serves only to prevent newcomers from activating until all currently active packets have transmitted successfully.

The busy tone yields a ***batch invariant***: there is only one batch running in the system at a time, which allows the throughput guarantees of the batch protocol to extend to arbitrary arrivals. Unfortunately, this basic protocol yields an unacceptable number of access attempts—one attempt per active packet per time step due to the busy tone. But even this primordial protocol is interesting because it shows a simple strategy for achieving constant throughput, in contrast to classical exponential backoff; see Figure 3.

We require a cheaper busy tone. A natural approach is for active packets to broadcast randomly on the control channel. This modified protocol broadcasts less, but it suffers the occasional ***control failure***, where the busy tone disappears even though some packets are still active.

The question is how active packets should respond to control failures. A plausible approach is to reset *every* packet, making every packet in the system restart in a single new batch. With no disruptions, this new protocol achieves constant throughput with a polylogarithmic number of access attempts.

But it is not robust to disruptions. The adversary has too much control: it can spoof the busy tone until packets have backed off a lot. The adversary then stops, and now packets have a very low probability of making an access attempt before a control failure causes a reset, which forces packets to join a new batch. Using this strategy, the adversary can prevent almost all of the packets in each batch from broadcasting successfully, forcing them to reset many times. Specifically, the adversary can keep packets in the system for $T \gg n$ time steps, and it can force $T^{\Theta(1)}$ access attempts rather than $\text{polylog}(n+T)$, access attempts, as specified by Theorems 1.1 and 1.2.

RE-BACKOFF addresses the previous concern by avoiding immediate resets; a packet only resets once a constant fraction of slots during its current batch are empty. Intuitively, the reset condition means that any packet that was reset could easily have chosen one of the empty slots, and just got unlucky. Any packet that enters a batch has at least a constant probability of broadcasting successfully in that batch and at most a constant probability of resetting. Therefore, in RE-BACKOFF a packet joins an expected constant number of batches before succeeding.

And so RE-BACKOFF sacrifices the batch invariant; multiple batches may exist in the system simultaneously and, consequently, we have put the throughput guarantee in jeopardy. This is because batch protocols do not perform well with dynamic arrivals. Even exponential backoff, which is already suboptimal on batches, performs asymptotically worse under dynamic arrivals.

The probabilistic busy tone and delayed reset serve together as a "leaky-mutual-exclusion" protocol, which keeps out many overlapping batches, but allows others to "leak" into the critical section. This contrasts with the (error-free) busy tone and aggressive reset mechanisms, each of which deterministically ensures

[2]Busy tones are also used in mutual exclusion and MAC protocols (see, e.g., [33, 58]) for coping with hidden terminal effects.

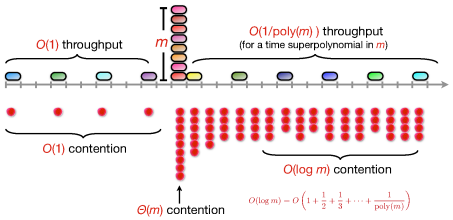

Figure 3: Illustration of how exponential backoff struggles with bursts [5]. Shown above the line are packet arrivals, and below the line are transmission attempts. (When there is more than one red dot below the line, there is a collision.) The packet arrivals illustrated comprise a steady stream of one new packet every three time steps, plus a single burst of m packets. Initially, the throughput is great. After the burst, the contention grows large, and the steady stream is enough to maintain the contention at $\Omega(\log(m))$ and reduce the throughput to $O(1/\text{poly}(m))$ for a length of time that is super polynomial in m.

mutual exclusion.

Most of the technical contribution of our paper has to do with proving that despite leaky mutual exclusion, Re-Backoff still guarantees constant contention (sum of broadcast probabilities) a constant fraction of the time, and therefore ensures constant nonwaste (and constant throughput when at most a constant fraction of slots are jammed). The idea is to prove that there are enough prefixes of slots so that: if the contention is much more or much less than a constant for X slots, then there are $\Omega(X)$ slots where contention is $\Theta(1)$.

Digging deeper, the technical hurdle that contention arguments seem to have is that contention changes over time in ways that are hard to characterize. For example, if the contention in a given time slot comes from a small number of young packets, then it will drop quickly over time (unless another batch activates), whereas if the contention comes from a large number of older packets, then it will drop gradually. Thus, there is a funny and unpredictable way in which the contention changes as a function of time.

Besides its complexity, what makes this proof unusual to us is that we are deprived of some of our favorite tools: high-probability arguments e.g., using Chernoff bounds. This tool is denied to us because the bursts may be arbitrarily smaller than the number of packets ever to enter the system.

Ultimately, we have a rather simple protocol that maintains, at its core, exponential backoff—while at the same time delivering the three desirable properties: constant throughput, few attempts, and robustness.

5 Throughput and Waste Analysis

In this section, we analyze Re-Backoff, showing that it achieves at most constant waste in both the finite and infinite cases.

Let s_i^t be the age of packet i in slot t, i.e., the number of slots that it has been active. At time t, we define the ***contention*** to be $X(t) = \sum_i 1/s_i^t$, where we sum over all the active packets. Thus, the expected number of broadcasts on the data channel in slot t is $dX(t)$. For every slot t, we define the value σ_t to be the minimum age out of all active packets such that the following hold: (i) $\sum_{i:0<s_i^t\le\sigma_t} 1/s_i^t \ge X(t)/2$; (ii) $\sum_{i:s_i^t\ge\sigma_t} 1/s_i^t \ge X(t)/2$. That is, active packets with age $\le \sigma_t$ have at least half the contention, and active packets with age $\ge \sigma_t$ have at least half the contention. We call these two sets the ***young*** and ***old*** packets, respectively. Note that packets with age exactly σ_t are *both* young and old.

Lemma 5.1. *For all times t, σ_t is well-defined.*

Proof. Sort the packets by age so that $s_1 \le s_2 \le \ldots$. Let k be the minimum index such that $X(t)/2 \le \sum_{i=1}^{k} 1/s_i \le X(t)/2 + 1/s_k$. Let $\sigma_t = s_k$. Notice that the young packets have contention at least $X(t)/2$, and the old packets have contention at least $X(t)/2$. □

We say that a ***control failure*** occurs in slot t if no packet broadcasts on the control channel during the slot. Recall that (1) a packet activation can occur only immediately after a control failure and (2) a packet j ***resets*** at time t if t is the first slot during j's lifetime $[t - s_j^t, t]$ of s_j slots, for which at least γs_j slots are empty.

Overview. In Section 5.1, we relate performance to contention. The tricky part is to bound *how often* and *for how long* the contention stays high. In Section 5.2, we break the execution up into epochs, structuring the changes in contention. We can then analyze the control failures (Section 5.3) and resets (Section 5.4) as a function of contention. This leads to a key result (Corollary 5.2) in Section 5.5 that shows that an epoch is "good" (in some sense) with constant probability.

One tricky aspect remains: the adversary may use the results from earlier epochs to bias later epochs by injecting new packets at just the wrong time. We introduce a simple probabilistic game, *the bad borrower game*, to capture this behavior and show that it cannot cause much harm (in Sections 5.6–5.8). Finally, we assemble the pieces in Sections 5.9 and 5.10, showing that we achieve at most a constant-factor waste.

5.1 Individual Slot Calculations

The next two lemmas look at the probability of a broadcast as a function of the contention, first looking at successful broadcasts and then all broadcasts—even those that result in a collision. (The proofs make use of the fact that $d = 1/2$, specifically relying on the inequality $d \leq 1/2$.)

LEMMA 5.2. *For a given slot t in which there is no disruption, the probability that some packet successfully broadcasts at time t is at least $\frac{d\,X(t)}{e^{2d\,X(t)}}$.*

Proof. Packet j is successful with probability $\frac{d}{s_j}\prod_{i\neq j}(1-\frac{d}{s_i})$. At most one packet is successful, so the success events for each packet are disjoint. The probability that some packet succeeds is thus at least $\frac{d}{s_1}\prod_i(1-\frac{d}{s_i})+\frac{d}{s_2}\prod_i(1-\frac{d}{s_i})+\cdots+\frac{d}{s_k}\prod_i(1-\frac{d}{s_i}) = \left(\prod_i(1-\frac{d}{s_i})\right)\cdot\sum_j\frac{d}{s_j} \geq \frac{dX(t)}{e^{2dX(t)}}$. The denominator follows from the fact that $1-x \geq e^{-2x}$ for $0 \leq x \leq 1/2$, and hence $\prod_i(1-\frac{d}{s_i}) \geq e^{-2d\sum_i(1/s_i)}$. □

LEMMA 5.3. *The probability that some packet is broadcast (not necessarily successfully) in slot t is at least $1-e^{-d\,X(t)}$ and at most $1-e^{-2d\,X(t)}$. The probability of a collision in the slot is at most $(1-e^{-2d\,X(t)})^2$.*

Proof. The probability that no packets broadcast is $\prod_i\left(1-\frac{d}{s_i}\right) \leq e^{-d\,X(t)}$. Conversely, $\prod_i\left(1-\frac{d}{s_i}\right) \geq e^{-2d\,X(t)}$ by the fact that $1-x \geq e^{-2x}$ for $0 \leq x \leq 1/2$. Let p be the probability that a slot is full. Then, the probability of a collision in that slot is at most p^2. Since $p \leq 1-e^{-2d\,X(t)}$, the claim follows. □

5.2 Epochs, Streaks, and Interstitial Slots

An execution is divided into two types of periods: ***epochs*** and ***interstitial slots***.

DEFINITION 3. *Each time t_0 when a packet is activated, a new **epoch** begins (and any earlier epoch ends). When an epoch ends, either a new epoch begins (if a new packet is activated) or there is a gap between epochs called the **interstitial slots**. To describe the duration of an epoch, we have two cases.*

*If the contention is not too high at the start of an epoch, specifically, if $X(t_0) < 8$, then the epoch consists of a single timestep t_0. We call such an epoch a **unit epoch**.*

*If $X(t_0) \geq 8$, then the epoch is subdivided further into a sequence of **streaks**, with the first streak beginning at t_0. If a streak begins at time t, then it ends at time $t' = t+\sigma_t$ (or at the start of a new epoch, whichever occurs first). If $X(t') < 8$, then the epoch ends. Otherwise another streak begins at time t' and ends at time $t'+\sigma_{t'}$.*

In general, we say that an epoch is ***disrupted*** if at least $1/4$ of its slots are disrupted. We next bound the change in contention during a streak.

LEMMA 5.4. *Assume that some streak begins at time t and that no control failures occur during the streak. Then $X(t+\sigma_t) \leq 3X(t)/4$.*

Proof. During the streak, all the young packets at time t *at least* double in age (since they each have age at most σ_t), so their contention reduces by at least half. Moreover, by definition the young packets at time t have contention at least $X(t)/2$, so the total contention reduces by at least $X(t)/4$. Since there are no control failures, there are no new packets activated and hence no increase in contention. □

LEMMA 5.5. *Assume that some streak begins at time t, where $X(t) \geq 8$, and that no resets occur during the streak. Then for all $t' \in [t, t+\sigma_t]$, $X(t') \geq X(t)/8$. More strongly, this $X(t)/8$ is a lower-bound on the contention contributed by just the old packets.*

Proof. Since there are no resets (and no activations, by definition) during the streak, the contention only decreases due to packets completing and due to increasing age. Consider the old packets at time t (which have contention at least $X(t)/2$ at time t). Since each of these packets at most doubles in age (since they have age $\geq \sigma_t$), their total contention would remain at least $X(t)/4$ throughout the streak.

Some of these packets may finish, thus reducing the contention further. Assume that the old packet with the largest contention completes in every slot—notice that each such packet that finishes reduces the contention by at most $1/\sigma_t$. Thus, if one such packet finishes in each slot of the streak, the total contention is reduced by at most $\sigma_t/\sigma_t = 1$. Thus, throughout the streak, the total contention from old packets remains at least $X(t)/4 - 1 \geq X(t)/8$ (since $X(t) \geq 8$). □

COROLLARY 5.1. *Consider any time t' that is part of an epoch. If no packets reset during the epoch before time t', then $X(t') \geq 1$.*

Proof. If t' is the first step of the epoch, then $X(t') \geq 1$ trivially (the packet whose activation started the epoch contributes 1 to contention).

Otherwise, there exists some time $t \leq t'$ that marks the start of the current streak. Since there have been no resets, Lemma 5.5 implies $X(t') \geq X(t)/8$. By definition of epochs/streak, $X(t) \geq 8$. We conclude that $X(t') \geq 1$. □

5.3 Control Failures

Next we look at the probability of a control failure as a function of contention. The next lemma argues that for the next slot in a streak (that has not yet had any failures), the old packets provide enough contention to make a control failure in the next slot unlikely. The subsequent lemma takes a union bound over all slots in the streak to conclude that it is unlikely for any control failure to occur in the streak.

LEMMA 5.6. *For a fixed time t when a streak begins, consider a control slot at time t' during the streak. Assume that there are no control failures or resets during the streak prior to time t'. Then the probability of a control failure in slot t' is at most $(\sigma_t)^{-\frac{cX(t)}{4}}$.*

Proof. The probability of a control failure in slot t' is at most (see Figure 2):

$$\begin{aligned}\prod_i \left(1 - \frac{c \ln s_i^{t'}}{s_i^{t'}}\right) &\leq e^{-c\sum_i \frac{1}{s_i^{t'}} \ln s_i^{t'}} \\ &= \sigma_t^{-c\sum_i \frac{\ln s_i^{t'}}{s_i^{t'} \ln \sigma_t}} \\ &\leq \sigma_t^{-c\sum_{\text{old } i} \frac{1}{s_i^{t'}}} \\ &\leq \sigma_t^{-cX(t)/8}\end{aligned}$$

The third line follows from the fact that $\ln s_i^{t'} \geq \ln \sigma_t$ for all old packets. The fourth line follows from Lemma 5.5. Note that the final result is a function of t, not t'. □

LEMMA 5.7. *For a fixed time t, consider a streak beginning at time t. Assume that no reset occurs during the streak. For any target $b > 0$, there exists a sufficiently large choice of algorithmic constant c such that the following holds: the probability that a control failure occurs in the interval $[t, t + \sigma_t]$ is at most $(\sigma_t)^{-bX(t)}$.*

Proof. Assuming there are no control failures during time $[t, t']$ for $t' \leq t + \sigma_t$, the probability of a control failure at time t' is at most $(\sigma_t)^{-\frac{cX(t)}{8}}$ by Lemma 5.6. Taking a union bound over the σ_t time slots, the probability of a control failure happening in any time slot is at most $(\sigma_t)^{-\frac{cX(t)}{8}+1}$. □

5.4 Bounding Resets

We next bound the probability that a reset takes place during an epoch. We show that with constant probability, a packet does not reset during an epoch; this is true since for any prefix of the epoch, there are sufficiently many broadcasts to prevent a reset. We first look at an abstract coin flipping game:

LEMMA 5.8. *Consider a sequence of independent Bernoulli trials each with probability (at least) p of success. Let $q = p^{16/p}$ be (a lower bound on) the probability that the first $16/p$ trials are all successful. Then with probability at least $q/2$: for all i, the first i trials contain at least $ip/4$ successes.*

Proof. Break up the trials into geometrically increasing subsequences of 2^k trials each. We say that a "failure" occurs in the kth subsequence if there are fewer than $p2^k/2$ successful trials within that subsequence. Using a Chernoff bound, the failure probability is at most $e^{-p2^k/8}$. Using a union bound, the probability that a failure occurs in any subsequence $k \geq \lg(1/p) + 4$ is at most $\sum_{k=\lg(1/p)+4}^{\infty} e^{-p2^k/8} = \sum_{j=1}^{\infty} e^{-2^j} \leq 1/2$. Therefore, with probability at least $q \cdot (1/2)$, the first $16/p$ trials are a success *and* every subsequence has at least $p2^k/2$ successes.

Now, suppose there is no failure in any subsequence, i.e., each has at least $p2^k/2$ successful trials. Pick any cutoff point in the subsequence of size 2^i and examine the total of 2^{i+1} trials up to this point. The previous subsequences for $k = 1, \ldots, i-1$ each contain at least $p2^k/2$ successful trials, for a total of at least $p \cdot 2^i/2$ successful trials. Therefore, up to the cutoff point, at least a $p/4$-fraction of the trials are successful. □

We conclude in the next lemma that the probability that any packet resets during the epoch is low. Note that the probability q in the following depends only on algorithmic parameter d, which is fixed at $d = 1/2$.

LEMMA 5.9. *There exists a constant $q > 0$ such that: the probability that* any *packet resets during a particular epoch is at most $1 - q/2$.*

Proof. Consider an execution of the protocol. Let t be the start of the epoch in question, and suppose that no packets reset before time $t' \geq t$ belonging to the same epoch. Then by Corollary 5.1, we have $X(t') \geq 1$. Combining this fact with Lemma 5.3, we have that slot t' is empty with probability at most $e^{-dX(t')} \leq e^{-d}$, i.e., it is full with probability at least $1 - e^{-d}$.

We shall now map the execution to the game in Lemma 5.8, beginning from the start of the epoch at time t. A slot $t' \geq t$ corresponds to a "success" if it is full and "failure" otherwise. The game continues until either a packet resets (we lose) or the epoch ends (we win). We have already argued that until we lose, we have success/full-slot probability of at least $p = 1 - e^{-d}$. We claim also (proof to follow) that we lose (a reset occurs) only if there exists an i such that the first i slots of the epoch contain less than $ip/4$ successes. Assuming the claim, we can apply Lemma 5.8 to conclude that we win with probability at least $q/2$ for $q = (1 - e^{-d})^{16/(1-e^{-d})}$.

To prove the claim, observe that no packets activate during the epoch, since activation starts a new epoch. Thus, any packets active at time t' must observe all slots in $[t, t']$. Let i be the number of slots in this interval. If the interval $[t, t']$ has at least $(1-\gamma)i = i/16$ successes, then no packets reset at time t' — a packet can only be below threshold at time t' if it was also below threshold at time t, which means it would have reset sooner. With $p = 1 - e^{-d} = 1 - e^{-1/2} > 1/4$, we have that a reset occurs only if there are less than $i/16 < ip/4$ successes, as claimed above. □

5.5 Successful Streaks

The notation $S(t)$ refers to a streak beginning at time t and continuing for σ_t slots. A streak is ***successful*** if there are no resets or packet activations during the streak. Notice that during a successful streak, we know that the contention is always at least $X(t)/8$ (by Lemma 5.5), and that at the end of the streak it is at most $3X(t)/4$ (by Lemma 5.4). We say that successful streaks $S(t)$ and $S(t')$ are consecutive if $t' = t + \sigma_t$.

The following lemma says that the length of any given streak dominates the sum of the lengths of all previous streaks in the epoch.

LEMMA 5.10. *Let $S(t_1), ..., S(t_k)$ be a set of consecutive streaks for which there is non-zero contention over each streak. Then, for $j = 2, ..., k$, $\sigma_{t_j} \geq \sum_{i<j} \sigma_{t_i}$.*

Proof. Starting from slot t_1, consider the set of active packets after $\sum_{1\leq i<k} \sigma_{t_i}$ slots for any $k \leq j$. Since the contention is non-zero, there exist remaining active packets. Moreover, since there are no injections over successful streaks, each packet has an age of at least $\sum_{1\leq i<k} \sigma_{t_i}$. Therefore, by the definition of σ_{t_k}, we have $\sigma_{t_k} \geq \sum_{1\leq i<k} \sigma_{t_i}$ for any $k \leq j$. □

We next prove a key result: an epoch is "successful" with constant probability. We analyze the change in contention and use the union bound over the streaks.

LEMMA 5.11. *For a non-unit epoch beginning at time t, with constant probability, every streak is successful.*

Proof. By assumption that the streak is non-unit, we have $X(t) \geq 8$. Let $S(t_1), ..., S(t_k)$ be consecutive streaks such that k is the first index in these consecutive streaks where the contention drops below 8. Define $\tau_j = \sum_{i\leq j} \sigma_{t_i}$.

For the special case where we might have $\sigma_{t_1} = 1$, the probability of a control failure in that single slot is at most $e^{-cX(t)} \leq e^{-8c}$ since $X(t) \geq 8$. This probability can be made as small as we desire by setting the appropriate value of c. For all other streaks, $\sigma_{t_i} \geq 2$.

By Lemma 5.7, the probability of a control failure over the interval $[t, t + \tau_k]$ is at most:

$$\max\left\{\left(\frac{1}{e}\right)^{c\cdot X(t)}, \left(\frac{1}{2}\right)^{b\cdot X(t)}\right\} + ... + \left(\frac{1}{2}\right)^{b\cdot X(t+\tau_{k-3})} + \left(\frac{1}{2}\right)^{b\cdot X(t+\tau_{k-2})} + \left(\frac{1}{2}\right)^{b\cdot X(t+\tau_{k-1})} + \left(\frac{1}{2}\right)^{b\cdot X(t+\tau_k)}.$$

By Lemma 5.4, the contention decreases by at least a multiplicative factor of 3/4 from one streak to the next. Additionally, the contention at the beginning of any streak is at least 8. Therefore, the contention decreases by at least an additive value of two when going from one streak to the next.

Returning to the above sum, the largest summand is the last one since this is when the contention is smallest. The summand to the left is at least a factor of 4 smaller, by our observation that the contention is decreasing by at least 2. Similarly, the term to the left of that is at least another factor of 4 smaller, and so on. Therefore, the value of the sum is upper-bounded by a geometric series, and so it is no more than $2 \cdot (1/2)^{b\cdot X(t+\tau_k)} \leq 2 \cdot (1/2)^{b\cdot 8}$ which we can set to any desired (small) constant δ depending only on a sufficiently large constants b and c.

By Lemma 5.9, the probability of a restart is at most a constant $q/2$ depending only on d. Therefore, the epoch is successful with probability at least $1-((1-\frac{q}{2})+\delta) = q/2-\delta \geq \varepsilon$ for some constant $\varepsilon > 0$ depending only on c and d. □

We say that an epoch is *disrupted* if at least 1/4 of the slots in the epoch are disrupted. The following corollary shows that we get constant throughput in an epoch with constant probability.

COROLLARY 5.2. *For a unit epoch that is not disrupted, with constant probability a packet broadcasts. For a non-unit epoch with length T, with constant probability: (i) every streak in the epoch is successful; (ii) the last streak is of length at least $T/2$; (iii) the contention throughout the last streak is between 1 and 8; and (iv) if the epoch is not disrupted, then at least $\Omega(T)$ packets are broadcast.*

Proof. Conclusion (i) follows from Lemma 5.11; conclusion (ii) follows from Lemma 5.10. Conclusion (iii) follows because there are no resets or packet activations; hence the contention decreases by at most a factor of 8 by Lemma 5.5. Conclusion (iv) follows from (ii) and (iii), and from observing that, in a non-disrupted slot in the last streak, there is a constant probability that a packet is broadcast, and a constant probability that one is not (due to an empty slot or a collision). If at most

1/4 of the epoch is disrupted, then at most half of the slots in the last streak are disrupted, and of these $T/4$ non-disrupted slots in the last epoch, in expectation, only a constant fraction are *not* successful broadcasts. Thus, by Markov's inequality, with constant probability, at most a constant fraction of these $T/4$ slots are not successful broadcasts, and hence with constant probability, at least $\Omega(T)$ packets are broadcast. □

5.6 Bad Borrower Game

We have shown that each epoch is ***good*** (satisfying Corollary 5.2) with constant probability. We now abstract away details, defining a simple game between two players: the ***lender*** and the ***borrower***. There are two key parameters: a probability p and a fraction $\alpha \in (0, 1)$. The game proceeds in iterations, where in each, the borrower borrows an arbitrary (adversarially chosen) amount of money from the lender, at least one dollar. With probability p, at the end of the iteration, the borrower repays a fraction α of the money.

The correspondence to our situation is as follows: each iteration is associated with an non-disrupted epoch, the length of the epoch defines the money borrowed, and the number of successful broadcasts defines the money repaid. In a good epoch, which occurs with constant probability p, if the epoch is not disrupted, then we get constant throughput and hence the borrower is repaid an α fraction of her money. In a bad epoch, by contrast, we allow for the worst case, which is no money paid back at all (no throughput, all waste).

For the ***finite Bad Borrower game***, there is a predetermined maximum amount that the lender can be repaid: after the borrower has been repaid n dollars, the game ends. This corresponds to a finite adversary that injects exactly n packets. In the ***infinite Bad Borrower game***, the game last forever, and an infinite amount of money is lent. This corresponds to infinite instances, where the adversary injects packets forever.

5.7 Finite Bad Borrower Game

Our goal in this section is to show that, when the borrower has repaid n dollars, he has borrowed at most $O(n)$ dollars. This corresponds to showing that n packets are successfully broadcast in $O(n)$ time, ignoring the interstitial slots (which we will come back to later). We assume throughout this section that n is the maximum amount of money repaid throughout the game, i.e., the adversary injects n packets in an execution. A simple correspondence lemma bounds the amount of money that the borrower can borrow:

LEMMA 5.12. *In every iteration of the finite bad borrower game, ≥ 1 dollar and $\leq n$ dollars are borrowed.*

Proof. The fact that the borrower borrows at least one dollar follows by definition. Assume the borrower borrows n dollars, i.e., that the associated epoch lasts for at least n slots. Recall that the last streak in the associated epoch must have been at least $n/2$ slots, and at the beginning of that final streak, the contention must have been at least 8 (or the epoch would have ended). Since the last streak is of length at least $n/2$, there must be a set of old packets with age $\geq n/2$ that collectively have contention at least 4 (by definition of a streak). This implies there must be at least $2n$ such packets, which is impossible, given the bound of n packets total. Thus, it is impossible to have an epoch of length n, and hence to borrow more than n dollars in an iteration of the finite bad borrower game. □

We now argue, via an analysis of the expected repayments, that when the finite bad borrower game ends, the expected cost to the lender is $O(n)$:

LEMMA 5.13. *Over an execution of the bad borrower game, the expected number of dollars borrowed is $O(n)$.*

Proof. We analyze the dollars repaid in the following fashion: we assume that for every dollar lent, it is paid back with probability $p\alpha$. Notice, of course, that these random choices are correlated: for a given iteration, either an α fraction of the dollars are paid back (with probability p), or no dollars are paid back, with probability $1-p$. For a given iteration of the game, if there are k dollars borrowed, we see that the expected number of dollars repaid is $pk\alpha$.

What is the expected number of dollars we have to lend in order for n dollars to be repaid? The answer is $n/(p\alpha)$, i.e., after $O(n)$ dollars have been borrowed, all n dollars have been repaid. In the last iteration, there can be at most n additional dollars lent (as part of the iteration where the last dollar is repaid), by Lemma 5.12, yielding an expected $O(n) + n$ borrowed dollars. □

5.8 Infinite Bad Borrower Game

In order to analyze infinite executions, we look at the infinite bad borrower game. Recall that, for parameter k chosen in advance, if there have been k dollars borrowed up to some point, then in expectation there have been $O(kp\alpha)$ dollars repaid. In the infinite case, we conclude something stronger: there are an infinite number of times where the borrower has repaid at least a $p\alpha/2$ fraction of the total dollars borrowed.

LEMMA 5.14. *For any iteration r of the infinite bad borrower game, with probability 1 there is a later iteration $r' > r$ such that if the lender has lent k dollars*

through iteration r', then the borrower has repaid at least $kp\alpha/2$ dollars.

Proof sketch. We can look at the random process as a one-dimensional biased random walk with variable step size. Let X_r be the value of the random walk in iteration r, where $X_0 = 0$. Our goal is to construct a random walk such $X_r = 0$ when exactly a $p\alpha/2$ fraction of borrowed money has been repaid, and $X_r > 0$ when more than enough has been repaid. To do so, we renormalize the random walk so that every dollar paid back moves a distant of $2/(p\alpha)$ to the right.

We thus define the random walk as follows. Suppose that we lend x dollars in iteration r. Then with probability p, the iteration is good (an α fraction of the borrowed money has been repaid) and we have $X_r = X_{r-1} - x + (x\alpha)(2/(p\alpha)) = X_{r-1} - x + 2x/p$. With probability $(1-p)$, the iteration is bad and we have $X_r = X_{r-1} - x$. Observe that the expected value of X_r is $X_{r-1} + x$.

It can be shown that the random walk is positive infinitely often, which is what is required to complete the proof. (Note that we cannot say that the random walk eventually remains always positive from some point on, which would be true of a biased random walk with fixed step size. The issue is that the adversary can always introduce very large steps, for example, by always borrowing more than X_{r-1} dollars in step r.) □

As a corollary, if we only consider the epochs, ignoring the contribution from the interstitial slots, we can show constant throughput for infinite executions. Specifically, we can use the infinite bad borrower game to define "measurement points," thus showing that in an infinite execution, there are an infinite number of points at which we get constant throughput (if we ignore the contribution from the interstitial slots).

COROLLARY 5.3. *If we take an infinite execution and remove all slots that are not part of an epoch, then the resulting execution has at most a constant fraction of waste.*

We later show (Section 5.10) that the contribution from the interstitial slots does not hurt the waste, meaning that we get at most a constant factor of waste taking into account all slots.

5.9 Interstitial Slots, Expected Waste, and Expected Throughput for Finite Instances

We begin by considering the finite case where there are n packets injected. We bound the length of the interstitial slots, after which we prove at most an expected constant factor of waste.

We first argue that for a packet's lifetime, any prefix of at least two slots is at least a constant fraction full.

LEMMA 5.15. *Suppose that a packet u is active for the time interval $[t, s]$. Then for any time t' with $t < t' \leq s$, at least a $1-\gamma = 1/16$ fraction of the slots in the interval $[t, t']$ are full.*

Proof. The proof is by contradiction. Suppose that $[t, t']$ is strictly less than a $(1-\gamma)$-fraction full. Let $x = t'+1-t$ be the total number of slots in the interval, and let k be the number of full slots in the interval. Then we have $x > \frac{1}{1-\gamma}k = 16k$. Since $x > 16k$ is an integer, $x \geq 16k+1$. Thus, the subinterval $[t, t'-1]$ contains at most k full slots across $x - 1 \geq 16k$ slots, meaning that a reset would occur at or before time $t'-1$. □

Our next lemma extends the above argument to cover all interstitial slots. We would like to say that the first t timeslots of the entire execution include at most a γ-fraction of empty slots. This is not necessarily true — the first slot could be empty. The issue is that Lemma 5.15 does not apply to the first step of a packet's lifetime. But we can make a similar claim if we elide certain slots. We define a slot to be ***active*** if at least one packet is active, and we define a slot to be a ***quiet arrival*** if a packet activates but the slot is empty. The following lemma achieves our goal by ignoring slots with quiet arrivals.

LEMMA 5.16. *For any integer $t > 0$, consider the first t active time slots that are not quiet arrivals. At most a $\gamma = 15/16$-fraction of these slots is empty.*

Proof. The proof is by induction on t. For the base case, observe that a quiet arrival results in an immediate reset of that packet. Thus, the first time step in consideration is a step during which a packet arrives and the slot is full.

For the inductive step, we assume that the claim holds for all of the t first steps and argue that it holds at $(t+1)$th. Consider any packet u that is active at time $t+1$. Let $t_u \leq t+1$ be the step at which u's current lifetime began. We have two cases.

Case 1. If $t_u = t+1$, then the packet just activated. By assumption, this is not a quiet arrival, and hence the $(t+1)$th step is full. By inductive assumption, the first t slots are at most a γ-fraction empty. Concatenating these slots proves the claim.

Case 2: $t_u < t+1$. Consider the interval $[t_u, t+1]$. Since u is active for the entire interval and $t_u < t+1$, we apply Lemma 5.15 to conclude that at most a γ fraction of these slots are empty. Eliding the quiet arrivals (which are empty slots) only reinforces this claim. By inductive

assumption, at most a γ fraction of the slots up to $t_u - 1$ are also empty. Concatenating these two intervals proves the claim. □

Observe that Lemma 5.16 counts all of the active interstitial slots, as any quiet arrivals are by definition part of an epoch. We thus have a way of charging the empty interstitial slots against full slots, incurring at most a $1/(1-\gamma)$ cost.

Our goal now is to bound the number of full interstitial slots, specifically the non-disrupted slots. The main idea is to show that for each non-empty and non-disrupted slot, there is a constant probability of successful transmission. Thus after $O(n)$ such slots, in expectation, all the packets have broadcast.

LEMMA 5.17. *For a slot s, let $e_{\geq 2}$ denote the event where two or more packets are broadcast in s, and let $e_{=1}$ denote the event where one packet is broadcast in slot s. If s is an interstitial slot, then $Pr(e_{\geq 2}) = O(Pr(e_{=1}))$.*

Proof. The lemma follows immediately from Lemmas 5.2 and 5.3, since in interstitial slots, the contention is $O(1)$. □

LEMMA 5.18. *There are at most $O(n)$ full, non-disrupted interstitial, slots in expectation.*

Proof. By the time that there are n full slots that have successful transmissions, the execution is over. And if we condition upon a given slot being full, there is a constant probability of a successful transmission by Lemma 5.17. Thus it takes $O(n)$ such slots, in expectation, before all n packets have successfully transmitted. □

We can now prove our claims in Theorem 1.1 and Corollary 1.1 regarding expected waste and throughput:

LEMMA 5.19. *If the adversary injects n packets,* RE-BACKOFF *has at most an expected constant factor waste.*

Proof. We will argue that the expected number of slots for all the packets to finish is: $\mathrm{E}[T] = O(n + D)$ slots. We then observe that the expected nonwaste is $\mathrm{E}[\lambda] = \mathrm{E}[(n + D)/T]$, which by Jensen's inequality is at least a constant. Throughout the proof we consider only active slots. Reincorporating the inactive slots only increases the waste by a constant factor as a packet activates after seeing an inactive slot.

Let n_e denote the total number of slots over all non-disrupted epochs[3], let n_d denote the number of slots over all disrupted epochs, and let n_i denote the number of full non-disrupted interstitial slots. Again, let D denote the total number of disrupted slots.

Our goal is to bound the number of empty interstitial slots. Lemma 5.16 implies that, ignoring some empty epoch slots (namely, the quiet arrivals), at most a δ-fraction of the remaining slots are empty. In particular, the worst case occurs if we pessimistically assume all epoch slots are full, giving at most $O(n_e+n_d+n_i+D)$ empty interstitial slots.

By Lemma 5.13, the finite bad borrower game implies that the number of epoch slots in non-disrupted epochs required to complete all n packets is $O(n)$ in expectation, therefore, $E[n_e] = O(n)$. As for the interstitial slots, Lemma 5.18 shows that $E[n_i] = O(n)$. Therefore, among non-disrupted epochs and non-disrupted interstitial slots, we conclude that the expected number of slots required for all n packets to succeed is $O(n)$.

Finally, we count the number of disrupted slots. Since a disrupted epoch is one in which at least 1/4 of the slots are disrupted, there are clearly at most $O(D)$ disrupted epoch slots. Similarly, there are at most $O(D)$ disrupted, non-empty insterstitial slots. There are also at most $O(D)$ slots in which the control channel is disrupted (which can cause wasted time on the data channel if there are no active packets). Thus, there are at most $O(D)$ such slots otherwise unaccounted for.

Thus we conclude that there are, in expectation, $O(n)$ non-disrupted epochs and non-disrupted interstitial slots, at most $O(D)$ disrupted slots and disrupted epochs, and $O(n + D)$ empty slots. □

5.10 Interstitial Slots for Infinite Instances

We now show that the contribution from the interstitial slots does not hurt the throughput in infinite executions. To do so, we deterministically bound the contribution from the empty interstitial slots. We show that as long as we pick "measurement points" that are sufficiently large that from then on the non-empty interstitial slots do not hurt. We use the following well known facts about random walks [44].

FACT 5.1. *Suppose that we have a biased random walk on a line with fixed step size, where the probability of going right is at least p, the probability of going left is at most $1 - p$, and the step size right is δ_r and the step size left is δ_ℓ. Suppose that $p\delta_r > (1 - p)\delta_\ell$. Then if the random walk starts at the origin, the probability of returning to the original is some constant strictly less than 1.*

COROLLARY 5.4. *For any such biased random walk, there is a last time that the walk returns to the origin.*

[3]Recall that an epoch is non-disrupted if $< 1/4$ of its slots are disrupted.

We use Corollary 5.4 to bound the ratio of collisions to broadcasts in non-disrupted, non-empty interstitial rounds. From some point on, the number of collisions is always at most a constant factor of the number of broadcasts, and hence yields constant throughput. We again observe that the empty slots cannot hurt the throughput by more than a constant factor overall, because, by Lemma 5.16 at most a γ fraction of the slots in any prefix can be empty interstitial slots. Finally, we bound the disrupted interstitial slots by D. From this, we conclude that we achieve constant nonwaste and throughput, as claimed in Theorem 1.2 and Corollary 1.2:

LEMMA 5.20. *In an infinite execution,* RE-BACKOFF *achieves at most a constant-fraction of waste.*

Proof. For some slot t, let i_t^b be the number of successful broadcasts in non-disrupted interstitial slots prior to time t, and let i_t^c be the number of collisions in non-disrupted interstitial slots prior to time t.

We first argue that, with probability 1, from some point t onwards, for all $t' > t$: $i_t^c \leq O(i_t^b)$. Conditioned on the fact that there is at least one broadcast in an non-disrupted interstitial round, let p be the probability of a successful broadcast and $q = 1 - p$ be the probability of a collision. We know from Lemma 5.17 that $q = O(p)$.

Define the following random walk: with probability q take a step to the left of size 1, and with probability p take a step to the right of size $2q/p$. Since $p \cdot (2q/p) > q$, by Corollary 5.4 we know that from some point on, this random walk is always positive.

Let t be a time slot that is after the last point where the random walk crosses the origin. We can then conclude that $i_t^c < i_t^b \cdot (2q/p)$. Since $2q/p = O(1)$, we conclude that $i_t^c = O(i_t^b)$.

Finally, we analyze the throughput. Fix any time t. Let $\hat{t}$ be the smallest time after t where the random walk defined above is positive. According to Lemma 5.14, there is a time $t' > \hat{t}$ where we have achieved constant throughput during the non-disrupted epochs, i.e., a constant fraction of the slots in non-disrupted epoch are broadcasts. By the analysis of the random walk, we conclude that a constant fraction of the non-empty, non-disrupted interstitial rounds are broadcasts. By assumption, at most $O(D)$ slots are part of disrupted epochs, and there are at most D disrupted interstitial slots. There are at most D disrupted control slots (which may cause delays on the data channel if there are no active packets). Finally, by Lemma 5.16, at most γ of the data channel slots in the entire execution are empty interstitial slots. Putting these pieces together yields constant throughput overall. □

6 Analysis of the Number of Access Attempts

In this section, we analyze the number of access attempts. We show that in the absence of disruption, the number of broadcasts is small and the adversary requires a significant amount of disruption to cause even a small increase in the number of access attempts.

We first analyze how often a packet resets, showing that it is likely to succeed before it has a chance to reset. This ensures that a packet cannot be forced to make a large number of attempts via repeated resets.

LEMMA 6.1. *When a packet becomes active, it succeeds (instead of resetting) with constant probability at least* $1 - e^{-d/8}$.

Proof. In this proof, we grant the adversary even more power than given by the model—in each slot, the adversary is allowed to specify whether the slot is "covered", meaning that it is either disrupted or some other packet transmits. The only thing the adversary does not control is the packet in question.

Starting from the time the packet becomes active, we divide time into windows $W_0, W_1, W_2, W_3, \ldots$, where window W_i has length 2^i. Note that if the first slot is covered, the packet cannot possibly reset until time 16 or later, which more than subsumes window W_1. Similarly, if at least 1 slot is also covered in window W_1, then the packet cannot possibly reset until after window W_2. In general, if at least half the slots are covered in each of the windows $W_0, W_1, \ldots, W_{i-1}$, then the packet either stays alive through W_i, or it succeeds sometime before the end of W_i—it cannot reset. Our argument thus proceeds inductively over windows, stopping at the first window W_i that is not at least half covered. In window W_i, the packet transmits independently in each data slot with probability at least $d/2^{i+2}$, where d is a constant specified in the protocol. Thus, if at least 2^{i-1} of the slots in W_i are left uncovered, the packet has either succeeded earlier, or it succeeds in window W_i with probability at least $1-(1-d/2^{i+2})^{2^{i-1}} \geq 1-1/e^{d/8}$, which is constant. □

COROLLARY 6.1. *For any positive integer k, the probability that a packet resets k times is at most* $1/e^{\Theta(k)}$.

Proof. The only observation we need is that for a particular packet, each of its lifetimes are nonoverlapping. Thus, Lemma 6.1 bounds the probability of a reset for a given lifetime, and for each lifetime the probabilities are independent. Thus, each reset occurs with probability at most $1/e^{d/8}$, and hence the probability of k resets is at most $1/e^{kd/8} = 1/e^{\Theta(k)}$. □

We can now bound the total number of access attempts that a packet makes during the first t slots

after its arrival. If it has not yet reset by time t, it is easy to see that it has made $O(\log^2 t+1)$ access attempts in expectation—and we have show above that a packet is unlikely to reset too many times. This yields:

LEMMA 6.2. *In the first t slots following a packet's arrival, it makes $O(\log^2(t)+1)$ attempts in expectation.*

Proof. Consider any lifetime of the packet. The expected number of access attempts during a slot is equal to the packet's transmission probability, and hence the expected total number of access attempts is the sum of probabilities across all slots by linearity of expectation. The expected number of access attempts in a lifetime is thus at most $\sum_{s=1}^{t} \Theta(1+\ln s)/s = \Theta(\log^2 t+1)$, with the $\Theta(1+\ln s)$ arising from the higher transmission probability of $c\max(\ln s, 1)/s$ in control slots.

We now compute the expected number of access attempts made by the packet by using linearity of expectation across all lifetimes of the packet. In particular, the number of access attempts during the kth lifetime is 0 if the packet does not reset $k-1$ times, and hence applying Corollary 6.1 the expected number of access attempts of the kth lifetime is $1/e^{\Theta(k)} \cdot O(\log^2 t+1)$. Using linearity of expectation across all lifetimes, we get a total expected number of access attempts of $O(\log^2 t+1)\sum_{k=0}^{\infty} 1/e^{\Theta(k)} = O(\log^2 t+1)$. □

We can now prove our claims in Theorems 1.1 and 1.2 regarding the expected number of access attempts per packet. In the finite case, we have already bounded the expected length of the execution in Lemma 5.19. In the infinite case, we separately analyze the disrupted slots, the non-disrupted slots with young packets, and the non-disrupted slots with old packets, bounding the number of attempts.

LEMMA 6.3. *Consider the finite case, let n be the number of injected packets, and let D be the number of disrupted slots. Then the expected number of access attempts each packet makes is $O(\log^2(n+D)+1)$.*

Proof. Suppose the execution completes in time t. Then applying Lemma 6.2 yields an expected number of access attempts of $O(\log^2 t+1)$, as the packet must complete before the execution completes. Let T be the expected time of completion. From Markov's inequality, $t \geq T^i$ with probability at most $1/T^{i-1}$. Summing across all i, the expected number of access attempts becomes $O(\sum_{i=1}^{\infty}(1/T^i)(\log^2(T^i)+1)) = O(\log^2 T+1)\cdot \sum_{i=1}^{\infty}(i^2/T^i) = O(\log^2 T)$. Substituting in the expected makespan $T = O(n+D)$ from Lemma 5.19 concludes the theorem. □

LEMMA 6.4. *Consider any time t in the infinite case at which we have λ throughput, for constant λ. Let D_t be the total number of disrupted slots before t, and let η_t be the maximum contention prior to time t. Then the expected average number of access attempts per packet active prior to time t is $O(\log^2(\eta_t + D_t)+1)$.*

Proof. The analysis is split into two cases: either $D_t \geq \lambda t/2$, or $D_t < \lambda t/2$. The first case is easy—Lemma 6.2 states that the expected number of access attempts per packet is $O(\log^2 t+1) = O(\log^2 D_t+1)$.

Suppose for the remainder that $D < \lambda t/2$. Then we divide the analysis here into three parts: (A) the disrupted slots, (B) the non-disrupted slots for "young" packets, and (C) the non-disrupted slots for "old" packets. In parts A and B, we shall show that the expected number of access attempts per packet is at most $O(\log^2 D_t + \log^2 \eta_t) = O(\log^2(D_t+\eta_t))$, regardless of whether we have constant throughput. It is only in part C that we leverage the assumption that time t is a time at which we have constant throughput.

A. Consider a single lifetime of a specific packet. Our goal is to bound the number of access attempts by this packet during all D_t disrupted slots. The number of access attempts is maximized if all D_t slots occur as early as possible in the packet's lifetime, in which case the expected number of access attempts during disrupted slots (i.e., during the first D_t slots of its lifetime) is at most $O(\log^2 D_t+1)$ per lifetime (Lemma 6.2). As in Lemma 6.2, we then apply Corollary 6.1 and linearity of expectation to get an expected number of access attempts of at most $O(\log^2 D_t+1)\sum_{k=0}^{\infty} 1/e^{\Theta(k)} = O(\log^2 D_t+1)$.

B. Consider a specific packet. We say that the packet is young during the first η_t^2 steps of its lifetime, during which it makes $O(\log^2 \eta_t^2+1) = O(\log^2 \eta_t+1)$ access attempts in expectation (Lemma 6.2). Summing across all lifetimes as above, we conclude that the contribution for young packets is $O(\log^2 \eta_t+1)$.

C. If a packet is not young, i.e., if its age is at least η_t^2, we say that it is old. Unlike parts A and B which analyze on a per-packet basis, this part analyzes the number of access attempts in aggregate. For every non-disrupted slot, there are at most η_t packets in the system, and hence at most η_t packets are old. Each old packet transmits with probability at most $O((1+\ln(\eta_t^2))/\eta_t^2) = O(1/\eta_t)$, and hence the expected number of access attempts across all old packets in any slot is $O(1)$. Using linearity of expectation across all t slots, we have a total expected number of access attempts by old packets of $O(t)$.

Since we have constant throughput at time t, we know that at least λt slots are either disrupted or successful transmissions. There are at most $\lambda t/2$

disrupted slots by assumption, and hence there are at least $\lambda t/2 = \Omega(t)$ successful transmissions. We charge the $O(t)$ number of access attempts from old packets to these $\Omega(t)$ completed packets, for an additional $O(1)$ number of access attempts per successful packet. □

7 Synchronization: Reducing to One Channel

In this section, we describe how to transform the Re-Backoff algorithm so that it runs on a single channel. We first describe the modified algorithm. We then show that it maintains a synchronized view of the channel. Finally, we review the analysis and see how it has to be modified for this variant.

7.1 Modified Algorithm

We begin by describing the modified algorithm that uses only one channel.

As in Re-Backoff, packets are initially inactive. They monitor the channel and wait to hear *two* empty slots, immediately after which they become active. (By contrast, in Re-Backoff, a packet becomes active in the next slot when it hears one empty *control* slot.)

The main idea is that once a packet becomes active, it alternates executing *control slots* and *data slots*. (This idea gets refined below to synchronize packets so that they agree on the parity of the slots.) When a packet first becomes active, it treats its first active slot as a control slot. In that first control slot, it broadcasts *with probability* 1. It then proceeds to alternate data and control slots. From then on, as in Re-Backoff, a node with age s_u broadcasts in a control slot with probability $c \max\{\ln s_u, 1\}/s_u$, and in a data slot with probability d/s_u (terminating upon success)—where c and d are constants as before.

For example, if packet u is inactive and it observes slot s and $s+1$ to be empty, it becomes active in slot $s+2$. It treats $s+2$ as a control slot and sends a control signal (with probability 1). It then alternates, treating $s+3$ as a data slot, $s+4$ as a control slot, etc.

A packet calculates its age as follows: prior to its first active slot (which it designates a control slot), it sets its age to 1; immediately before every subsequent control slot, it increments its age. That is, the age of a packet in slot s is the number of slots that it has designated as control slots since it became active, up to and including slot s.

With no further synchronization, different packets may treat a given slot as a control slot and a data slot at the same time. We thus add a synchronization mechanism:

> If a packet observes an empty control slot followed by a non-empty data slot, then it designates the following slot as a data slot.

In doing so, it breaks the rule of alternation, synchronizing the packets.

For example, suppose packet u believes that s is a control slot. If s is empty (i.e., no broadcast, no collision, no disruption), and if $s+1$ is non-empty (i.e., a broadcast or a collision or disruption), then packet u treats slot $s+2$ as a data slot. It then continues to alternate, treating $s+3$ as a control slot.

If a packet completes in a data slot that immediately follows an empty control slot, then it does not terminate immediately, but participates in the additional data slot that follows before terminating.

Finally, recall that a packet resets if it finds that a γ-fraction of data slots have been empty since it became active. Here, the reset rule remain identical with one change: if there are two consecutive data slots, then the packet does not count the results from the first data slot.

For example, if s is a control slot and $s+1$ and $s+2$ are data slots (because s is empty and $s+1$ is non-empty), then after these three slots: if $s+2$ is empty, a packet increments its count of empty data slots by one; if $s+2$ is full, a packet increments its count of full data slots by one.

7.2 Slot Agreement

We observe that packets agree on whether a slot is a control or a data slot, i.e., synchronization works:

Lemma 7.1. *Let t be a slot, and let u and v be two packets that are active in slot $t-1$ and slot t. Then u considers t a control slot if and only if v considers t a control slot. Similarly, u considers t a data slot if and only if v considers t a data slot.*

Proof. Assume u and v are injected in the same slot $t_0 < t$. Then u and v both consider t_0 to be a control slot, and observe the same pattern of full and empty slots from then on. Hence u and v continue to identify slots in the same manner.

Assume instead that u is injected in slot t_u and v is injected in slot $t_v > t_u$ (where $t_v < t$). There are two cases. First, the slot t_v may be considered a control slot by u. In that case, as of slot t_v, both u and v consider t_v to be a control slot and both observe the same pattern of full and empty slots from then on. Hence u and v continue to identify slots in the same manner.

Finally, assume that u considers slot t_v to be a data slot. We know that v broadcasts in slot t_v, because a packet broadcasts in its first control slot, and hence t_v is non-empty. Notice, though, that $t_v - 1$ and $t_v - 2$ must be empty slots, as packet v only becomes active

after observing two empty slots. From this we conclude that u considers $t_v - 1$ to be a control slot: we know (by assumption) that t_v is a data slot; if $t_v - 1$ were also a data slot, then the preceding control slot $t_v - 2$ must also have been non-empty (in order to force the repeated data slot); however, we know that $t_v - 2$ is empty.

Since u considers $t_v - 1$ to be an *empty* control slot, and u considers t_v to be a non-empty data slot, it designated slots $t_v + 1$ to be a data slot. Packet v also considers $t_v + 1$ to be a data slot (because it is alternating slot types). Hence, as of slot $t_v + 1$, both u and v agree on the slot designation. From that point on, but packet u and v observe the same pattern of full and empty slots, and hence they continue to identify slots in the same manner. □

As a result of Lemma 7.1, we can officialy designate slots as control or data slots. Consider a slot t:

- If no packet is active in slot t, we designate slot t to be an empty slot.
- If there exists any packet u that is active in slot $t - 1$ and slot t, then we designate slot t a data or control slot based on the designation of packet u.
- If all the packets active in slot t were *not* active in slot $t-1$, then we designate slot t a control slot (as do all the newly activated packets).

We can then designate a pair or triple of slots consisting of a control slot followed by one or two data slots as a slot-group. Throughout, unless specified otherwise, when we refer to a slot as a control or a data slot, we refer to this global (synchronized) designation.

7.3 Changes to the Analysis

We can then repeat the analysis found in Sections 5 and 6, with a very small number of minor changes, yielding the same waste, throughput, and energy results. Here, we highlight some of these issues.

The analysis begins by defining contention, dividing packets into young and old packets, definiting control failures, etc. Nothing needs to change as a result. It is perhaps worth noting that a control failure is no longer a sufficient condition for a packet to become active (as two consecutive slots are needed), but it remains a necessary condition

We proceed to calculate the probability of various events as a function of contention (e.g., Lemma 5.2 and Lemma 5.3), which remain identical for all slots in which no packet becomes active.

We define epochs and streaks as before, i.e., an epoch begins at the beginning of a slot group in which some packet is activated, and a streak to consist of the subsequent σ_t slot-groups. If a streak ends with contention below 32, then the epoch ends and interstital slots begin; otherwise a new streak begins. (Notice that we have increased the contention limit for an epoch.)

We say say that an epoch is disrupted if at least 1/4 of its data slots are disrupted; for this purpose, if a slot-group contains three slots, we count only the second data slot.

The analysis of contention remains unchanged (e.g., Lemma 5.4 and Lemma 5.5), with the only difference that, due to the extra data slots, more packets can complete, lowering the contention twice as fast. Thus, we assume that initially $X(t) \geq 16$, and conclude that it does not drop by more than a factor of 16. Similarly, the analysis of control failures (Lemma 5.6 and Lemma 5.7) remains unchanged.

The analysis of resets, howevers, requires a small modification. Consider Lemma 5.9, which argues that a reset occurs during an epoch with at most constant probability, specifically, $1 - (q/2)$ for some constant $q > 0$ depending only on d.

Here, we need to modify the proof slightly. Consider a slot-group with an empty control slot. In this case, there are two cases that lead to us "counting" an empty data slot (and increasing the likilihood of reset):

- The data slot is empty.
- The data slot is non-empty, followed immediately by an empty data slot.

The first non-empty data slot triggers another data slot (for synchronization purposes), which may gives a second chance for an empty data slot. (Recall that we count a data slot as empty for a slot-group in a three-slot slot-group if the second data slot is empty.)

Thus, we modify the proof as follows. First, we consider epochs whose initial contention is 32. This ensures that through the epoch, the contention is at least 2, and hence the probability of an empty data slot is at most e^{-2d} (by Lemma 5.3).

As such, the probability that, for a given slot-group, we count a data slot as empty is at most $2e^{-2d}$. Hence, for the purpose of Lemma 5.9, we define $p = 1 - 2e^{-d}$. The remainder of the analysis continues as before, with the conclusion that at least $p/4 \geq 1 - \gamma$ fraction of the slots in the epoch are full with probability at least $q/2$.

The remainder of the throughput analysis continues unchanged. For example, Lemma 5.11 is a critical lemma that shows that every streak is successful with constant probability. This follows from a calculation based on the probability of control failures and resets. As those bounds remain unchanged, the lemma holds unchanged here. As a result, Corollary 5.2 also holds

unchanged, where lengths refer to the number of slot-groups (instead of the number of slots).

Similarly, the analysis of the bad borrower game (Lemma 5.12, Lemma 5.13, Lemma 5.14, Corollary 5.3) are not impacted by the modifications.

It remains to consider the interstitial slots. Lemma 5.15 shows that for any prefix of a packet's lifetime (as long as it is length at least 2), at least a $(1-\gamma)$ fraction of the data slots are full. This remains true, so long as we count only the second data slots in a 3-slot slot-group.

Lemma 5.16 then follows, implying that for timeslots that are non-quiet arrivals, at most a γ-fraction of the data slots are empty. Again, this holds identically, so long as we only count the second data slot in a 3-slot slot-group. (Moreover, the first data-slot in a 3-slot slot-group is always non-empty, by definition.)

At this point, we have accounted for all the empty interstitial slots and the remaining accounting continues as before. We conclude Lemma 5.18 excactly as before, i.e., there are at most $O(n)$ full non-disrupted interstitial slots. Putting the pieces together, this yields Theorem 1.1 which shows expected constant throughput in finite executions.

The analysis of interstitial slots in the infinite case proceeds similarly, with Lemma 5.20 following as before from the same basic analysis of random walks (which does not depend on the protocol itself, and hence is unchanged).

Finally, the energy analysis is unchanged, beyond accounting for the energy used in the extra data slots. A packet may now spend somewhat more energy, as it may now broadcast in two data slots for some slot-groups instead of just one. This change, however, increases the energy usage by at most a constant factor.

References

[1] David J. Aldous. Ultimate instability of exponential back-off protocol for acknowledgment-based transmission control of random access communication channels. *IEEE Transactions on Information Theory*, 33(2):219–223, 1987.

[2] Lakshmi Anantharamu, Bogdan S. Chlebus, and Mariusz 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.

[3] Baruch Awerbuch, Andrea Richa, and Christian 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] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, September 1999.

[5] Michael A. Bender, Martin Farach-Colton, Simai He, Bradley C. Kuszmaul, and Charles 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] Michael A. Bender, Jeremy T. Fineman, and Seth Gilbert. Contention resolution with heterogeneous job sizes. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA)*, pages 112–123, 2006.

[7] Petra Berenbrink, Artur Czumaj, Matthias Englert, Tom Friedetzky, and Lars Nagel. Multiple-choice balanced allocation in (almost) parallel. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX-RANDOM)*, pages 411–422, 2012.

[8] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold Vöcking. Balanced allocations: The heavily loaded case. *SIAM J. Comput.*, 35(6):1350–1385, 2006.

[9] Petra Berenbrink, Kamyar Khodamoradi, Thomas Sauerwald, and Alexandre 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.

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

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

[12] John I. Capetanakis. Generalized TDMA: The multi-accessing tree protocol. *IEEE Transactions on Communications*, 27(10):1476–1484, Oct 1979.

[13] Bogdan S. Chlebus, Leszek Gasieniec, Dariusz R. Kowalski, and Tomasz 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] Bogdan S. Chlebus and Dariusz 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] Bogdan S. Chlebus, Dariusz R. Kowalski, and Mariusz A. Rokicki. Adversarial queuing on the multiple-access channel. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 92–101, 2006.

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

[17] Marek Chrobak, Leszek Gasieniec, and Dariusz R. Kowalski. The wake-up problem in multihop radio networks. *SIAM Journal on Computing*, 36(5):1453–1471, 2007.

[18] Richard Cole, Alan M. Frieze, Bruce M. Maggs,

Michael Mitzenmacher, Andréa W. Richa, Ramesh K. Sitaraman, and Eli Upfal. On balls and bins with deletions. In *Randomization and Approximation Techniques in Computer Science, Second International Workshop (RANDOM)*, pages 145–158, 1998.

[19] Bryan Costales and Eric Allman. *Sendmail*. O'Reilly, third edition, December 2002.

[20] Antonio Fernández Anta, Miguel A. Mosteiro, and Jorge Ramón Muñoz. Unbounded contention resolution in multiple-access channels. *Algorithmica*, 67(3):295–314, 2013.

[21] Mihály Geréb-Graus and Thanasis 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 and P.D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. ALCOM-IT Technical Report TR-074-96, Warwick, 1996. `http://www.dcs.warwick.ac.uk/~leslie/alcompapers/contention.ps`.

[23] Leslie Ann Goldberg, Mark Jerrum, Tom Leighton, and Satish Rao. A doubly logarithmic communication algorithm for the completely connected optical communication parallel computer. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 300–309, 1993.

[24] Leslie Ann Goldberg, Mark Jerrum, Tom Leighton, and Satish Rao. Doubly logarithmic communication algorithms for optical-communication parallel computers. *SIAM Journal on Computing*, 26(4):1100–1119, August 1997.

[25] Leslie Ann Goldberg and Philip D. MacKenzie. Analysis of practical backoff protocols for contention resolution with multiple servers. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 554–563, 1996.

[26] Leslie Ann Goldberg, Philip D. MacKenzie, Mike Paterson, and Aravind Srinivasan. Contention resolution with constant expected delay. *Journal of the ACM*, 47(6):1048–1096, November 2000.

[27] Leslie Ann Goldberg, Philip D. Mackenzie, Mike Paterson, and Aravind Srinivasan. Contention resolution with constant expected delay. *Journal of the ACM*, 47(6):1048–1096, 2000.

[28] Leslie Ann Goldberg, Yossi Matias, and Satish Rao. An optical simulation of shared memory. *SIAM Journal on Computing*, 28(5):1829–1847, October 1999.

[29] Jonathan Goodman, Albert G. Greenberg, Neal Madras, and Peter March. Stability of binary exponential backoff. *Journal of the ACM*, 35(3):579–602, July 1988.

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

[31] Albert G. Greenberg, Philippe Flajolet, and Richard E. Ladner. Estimating the multiplicities of conflicts to speed their resolution in multiple access channels. *JACM*, 34(2):289–325, April 1987.

[32] Albert G. Greenberg and Shmuel 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.

[33] Zygmunt J. Haas and Jing Deng. Dual busy tone multiple access (DBTMA) - A multiple access control scheme for ad hoc networks. *Communications, IEEE Transactions on*, 50(6):975–985, Jun 2002.

[34] Johan Hastad, Tom Leighton, and Brian Rogoff. Analysis of backoff protocols for multiple access channels. In *Proceedings of the Ninteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 241–253, New York, New York, May 1987.

[35] Johan Hastad, Tom Leighton, and Brian Rogoff. Analysis of backoff protocols for mulitiple access channels. *SIAM Journal on Computing*, 25(4):1996, 740-774.

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

[37] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A framework for classifying denial of service attacks. In *2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 99–110, 2003.

[38] Van Jacobson and Michael J. Karels. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols (SIGCOMM)*, pages 314–329, 1988.

[39] Marek Klonowski and Dominik Pajak. Electing a leader in wireless networks quickly despite jamming. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures*, pages 304–312, 2015.

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

[41] Byung-Jae Kwak, Nah-Oak Song, and Leonard E. Miller. Performance analysis of exponential backoff. *IEEE/ACM Transactions on Networking*, 13(2):2005, 343-355.

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

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

[44] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[45] Amit Mondal and Aleksandar Kuzmanovic. Removing exponential backoff from TCP. *SIGCOMM Comput. Commun. Rev.*, 38(5):17–28, September 2008.

[46] Adrian Ogierman, Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive MAC under adversarial SINR. In *Proceedings of the* 33^{rd} *Annual IEEE International Conference on Computer Communications (INFOCOM)*, pages 2751–2759, 2014.

[47] Google Apps 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.

[48] Prabhakar Raghavan and Eli Upfal. Stochastic contention resolution with short delays. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing (STOC)*, pages 229–237, 1995.

[49] Prabhakar Raghavan and Eli Upfal. Stochastic contention resolution with short delays. *SIAM Journal on Computing*, 28(2):709–719, April 1999.

[50] Ravi Rajwar and James R. Goodman. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *Proceedings of the 34th Annual Intnl. Symposium on Microarchitecture*, pages 294–305, Austin, Texas, December 2001.

[51] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin 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.

[52] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin 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.

[53] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin 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.

[54] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive throughput in multi-hop wireless networks despite adaptive jamming. *Distributed Computing*, 26(3):159–171, 2013.

[55] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. An efficient and fair MAC protocol robust to reactive interference. *IEEE/ACM Transactions on Networking*, 21(1):760–771, 2013.

[56] Andrea 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.

[57] Amazon Web Services. Error retries and exponential backoff in aws. `http://docs.aws.amazon.com/general/latest/gr/api-retries.html`, 2012.

[58] Cheng shong Wu and Victor O.K. Li. Receiver-initiated busy-tone multiple access in packet radio networks. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology*, pages 336–342, 1988.

[59] Nah-Oak Song, Byung-Jae Kwak, and Leonard E. Miller. On the stability of exponential backoff. *Journal of Research of the National Institute of Standards and Technology*, 108(4), 2003.

[60] Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003.

[61] John Paul Walters, Zhengqiang Liang, Weisong Shi, and Vipin Chaudhary. *Security in Distributed, Grid, Mobile, and Pervasive Computing. Chapter 17: Wireless Sensor Network Security: A Survey.* Auerbach Publications, 2007.

[62] Dan E. Willard. Log-logarithmic protocols for resolving ethernet and semaphore conflicts. In *Proceeedings of the 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 512–521.

[63] Yang 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.

[64] Wenyuan Xu, Wade Trappe, Yanyong Zhang, and Timothy Wood. The feasibility of launching and detecting jamming attacks in wireless networks. In *Proceedings of the 6th ACM International Symposium on Mobile ad hoc Networking and Computing (MobiHoc)*, pages 46–57, 2005.