Abstract

The point of adversarial analysis is to model the worst-case performance of an algorithm. Unfortunately, this analysis may not always reflect performance in practice because the adversarial assumption can be overly pessimistic. In such cases, several techniques have been developed to provide a more refined understanding of how an algorithm performs e.g., competitive analysis, parameterized analysis, and the theory of approximation algorithms.

Here, we describe an analogous technique called resource competitiveness, tailored for distributed systems. Often there is an operational cost for adversarial behavior arising from bandwidth usage, computational power, energy limitations, etc. Modeling this cost provides some notion of how much disruption the adversary can inflict on the system. In parameterizing by this cost, we can design an algorithm with the following guarantee: if the adversary pays $T$, then the additional cost of the algorithm is some function of $T$.

Resource competitiveness yields results pertaining to secure, fault tolerant, and efficient distributed computation. We summarize these results and highlight future challenges where we expect this algorithmic tool to provide new insights.
1 Introduction

In his influential exegesis on *The Art of War*, the warlord Cao Cao wrote that “he who wishes to fight must first count the cost”, noting that preparing for conflict requires a careful accounting of available resources [71]. In this article, we argue that this maxim should guide our approach to distributed computation, which is often analyzed as a struggle between an algorithm and an adversary.

Resource competitiveness is a recent algorithmic technique that accounts for the cost incurred by an adversary for disrupting the system. Here, the notion of cost corresponds to any resource such as bandwidth, computational power, or an onboard energy supply. In parameterizing by this cost, we can design an algorithm with the following guarantee: if the adversary pays $T$, then the additional cost of the algorithm is some function of $T$.

In much of the literature on robust distributed computing, the adversary has a known (or upper bounded) budget that can be used to disrupt a task being executed by correct nodes. As a common example, this budget may be expressed by the number of malicious (bad) nodes controlled by the adversary. In this context, a number of seminal results have originated from the theory community (for examples, see [29, 46, 57]) and from the community of practitioners (for examples, see [1, 16, 45, 60, 65]).

Such results fix the maximum amount of resources at the adversary’s disposal — expressed as an upper bound on the number of bad nodes, $t$, that can be tolerated — and then focus on optimizing metrics such as latency or communication overhead. There are many settings where this treatment makes sense: perhaps the adversary is unconcerned with its own costs, or the distributed computation is provably impossible beyond this maximum budget $t$. However, there are limitations inherent to this approach:

- Both the good nodes and the adversary may be resource constrained, and ignoring this aspect places algorithm designers at an unnecessary disadvantage. Instead, we should incorporate the resource constraints of the adversary into the design of our algorithms. Such a situation may arise when the adversary incurs a cost to obtain physical resources for launching attacks; for example, there are monetary costs for renting a botnet [30]. In such cases, a notion of relative cost is compelling.

- The budget may be unknown. Moreover, the adversary may never actually utilize any of this budget. Consequently, an algorithm that proactively has a large overhead to tolerate disruption — disruption which may never occur — is inefficient.

By ignoring these aspects, traditional approaches to fault tolerance consider a one-sided picture. Instead, a more complete approach follows from measuring the performance of an algorithm relative to the amount of disruption in the system.

1.1 A Formal Definition of Resource Competitiveness

We now formally define what it means for a distributed algorithm $\mathcal{A}$ to be resource competitive. This was originally proposed in [37], but we refine the definition here.
Assume a system with a set $G$ of $n$ good nodes that obey actions specified by algorithm $A$. There exists an adversary who incarnates a source of disruption in the system. For example, the adversary may represent (1) any number of malicious nodes that collude and deviate arbitrarily from $A$, or (2) the effects of more benign failures due to software or hardware faults.

Let $\text{Cost}(\alpha, v)$ denote the resource expenditure (or cost) to a good node $v$ for executing the actions prescribed by $A$ in an execution $\alpha$. A resource might be bandwidth, CPU cycles, energy, actual money, or another useful domain-specific measure.

Let $T(\alpha)$ be the adversary’s total cost; this is typically unknown to the good nodes. It is possible for $\text{Cost}(\alpha, v)$ and $T(\alpha)$ to correspond to different resources. For example, good nodes may be concerned with bandwidth while the adversary is concerned with CPU cycles.

We now define what it means for $A$ to be resource competitive:

**Definition 1.** Algorithm $A$ is $(\rho, \tau)$-resource competitive if $\max_{v \in G}\{\text{Cost}(\alpha, v)\} \leq \rho(T(\alpha)) + \tau$ for any $\alpha$.

Definition 1 states that $A$ is resource competitive if the maximum cost incurred by any good node is less than some function of the adversary’s total cost, $\rho(T(\alpha))$, plus some additive term $\tau$, where both $\rho$ and $\tau$ are functions mapping to non-negative real values. The function $\rho$ is called the robustness function and it is a function of $T$ and possibly other parameters such as $n$. Throughout, we will simply refer to $T$ instead of $T(\alpha)$ since $\alpha$ is implicit.

Why do we need $\tau$? Note that when $T = 0$ (there is no disruption), the good nodes clearly cannot incur less than zero cost; $\tau$ represents the unavoidable cost to attain a goal when there is no disruption. It is useful to make this separation explicit via this notation $\tau$ which we call the efficiency function; $\tau$ can be a function of parameters such as $n$, but it is not a function of $T$.

In designing $A$, we desire $\rho$ to be a slow-growing function. The efficiency function, $\tau$, may not be a slow growing function, but it should be small relative to the best-performing algorithm that functions in the absence of disruption. For example, if a distributed computation requires $n$ messages to be exchanged, then $\tau = O(n)$ would imply only a constant-factor increase in overhead which, while not a slow-growing function, is asymptotically optimal.

In the example to follow in Section 2, we will observe the following adaptive behavior of $A$. When $T$ is zero or “small”, the efficiency function will dominate the cost of $A$; this is the unavoidable (or upfront) cost of running the algorithm. As $T$ increases, the robustness function will be the dominating component. Therefore, beyond the upfront cost, the cost of $A$ grows as a function of the amount of disruption.

Functions other than the maximum cost over all nodes may be appropriate depending on the context. For instance, we may consider the average or median cost. Furthermore, if a resource-competitive algorithm is randomized, then we can speak of cost in terms of a high probability guarantee, or in expectation, with minor changes to the definition.

Definition 1 accounts for finite executions. In the case of an infinite execution, an algorithm is resource competitive if it is resource-competitive for every prefix of the execution. Finally, we note that resource-competitive results are typically reported using big-O notation of the form $O(\rho(T) + \tau)$. 
We now provide a concrete “proof of concept” by summarizing a result from [43] dealing with wireless sensor networks. The wireless medium is vulnerable to interference caused by malfunctioning, or even malicious, devices; such interference is called jamming.

A number of elegant results have addressed communication under jamming attacks (see [76] for examples). However, these prior works constrain the jamming strategy of the adversary in some fashion; for example, the jamming may be random [58], or bounded within a window time [5, 21, 61–63], or limited to a subset of the total spectrum [24, 25, 28, 35, 50].

Using a resource-competitive approach, we can avoid placing these constraints on the jammer. Instead, we should identify a relevant resource and model its expenditure. In many wireless network settings, there are stringent energy constraints placed on devices since they are typically battery powered. For example, a sensor network may be deployed in hard-to-access terrain, and once a device has exhausted its energy supply, it may be permanently disabled. The devices controlled by the adversary are also likely to be battery powered. Therefore, energy is a constrained resource for both good and bad devices.

How is usage of this resource charged? In practice, accessing the channel dominates the operational cost of a device. Therefore, we assume that sending, listening, and jamming are the costly operations in our model that incur a resource expenditure.

**Problem and Model:** Alice wants to send a message $m$ to Bob over a wireless channel despite a jamming adversary. We must guarantee that (1) Bob receives $m$, and (2) that Alice learns that Bob receives $m$.

Time is divided into discrete slots and $m$ is assumed to fit in a single slot (if not, $m$ can be sent piecewise). Messages from Alice can be authenticated and she cannot be corrupted. In practice, scalable dissemination of a small number of public keys is possible and we may assume that Alice’s public key is known to receivers in her broadcast range. However, we do not assume that Bob can be authenticated.

As stated earlier, jamming, sending, and listening on the channel is expensive, and we assign each operation a unit cost per slot. A slot where no node uses the channel is clear, while a slot where the adversary jams is jammed. A clear slot cannot be forged by the adversary (see [14]).
In any slot, Alice and Bob are assumed to be in the energy-efficient sleep state if they are not sending or listening. For simplicity, assume that, in any slot, the adversary’s decision to jam is made independently of Alice’s or Bob’s actions.\footnote{Under certain conditions, a stronger adversary that is adaptive — uses past information to inform its decision to jam in the current slot — or reactive — knows the actions of Alice and Bob in the current slot before deciding to jam — can be tolerated.}

**A Naive Approach:** Alice and Bob can try to outspend the adversary. For example, in each even-indexed slot, Alice sends $m$ while Bob listens. In each odd-indexed slot, if Bob has not received $m$, he sends a negative acknowledgement (\texttt{nack}) message; otherwise, Bob terminates. Alice listens in each odd-indexed slot and, if Alice receives a \texttt{nack}, she continues to the next even-indexed slot and sends $m$. Similarly, if Alice detects a blocked odd-indexed slot, she interprets this as the situation where Bob sent \texttt{nack} but the slot was jammed; therefore, she continues with the protocol. However, if Alice detects a clear odd-indexed slot, she knows that Bob received $m$ and terminated since the adversary cannot forge a clear slot; in this case, Alice safely terminates.

While Alice and Bob both finish correctly, note that if the adversary jams $T$ consecutive even-indexed slots, then Alice and Bob each send and listen for $2T + 2$ slots. Therefore, Alice and Bob spend more than twice what the adversary spends; that is, $\rho(T) > 2T$ and the adversary rapidly disables each node by depleting its energy supply. This illustrates why jamming can be an effective attack.

**The LESS NAIVE Algorithm:** A first attempt at a resource-competitive approach is LESS NAIVE in Figure 1. In the Send Phase, over each of $\ell$ slots, Alice and Bob are probabilistically sending and listening, respectively, with probability $d/\sqrt{\ell}$ where $d > 0$ is a constant. If Bob ever receives $m$, he terminates the protocol. In the Nack Phase, if Bob has not terminated, he sends \texttt{nack} to Alice during the single slot asking her to enter into a new round. If Alice hears \texttt{nack}, or if the slot is jammed, she proceeds into the next round; otherwise, if the slot is clear, she terminates.

Using a birthday-paradox-like argument, there is likely to be a non-jammed slot where both Alice sends $m$ and Bob listens; indeed, for an appropriately chosen $d$, this is true even if up to $\ell/2$ slots in the Send Phase are jammed. Therefore, communication is likely to succeed unless the adversary jams more than half the slots. Conversely, when more than half the slots are jammed, Bob may not receive $m$. But now $T = \Omega(\ell)$ while both Alice and Bob spend only $O(\sqrt{\ell}) = O(\sqrt{T})$ in expectation. Therefore, to prevent communication, the adversary must incur a cost that is roughly quadratically larger. This is exactly the flavor of result that we seek.

Are we done? No, LESS NAIVE is vulnerable to the following attack. Assume that Bob receives $m$ and terminates. Then, in the next Nack Phase, the adversary may spoof a \texttt{nack} message and force Alice to execute another round. In each subsequent Send Phase, the adversary will sleep and then awaken in the Nack Phase to send \texttt{nack} again. Even if messages from Bob can be authenticated, the adversary can simply generate noise which will yield the same result; this is unavoidable since collision detection is used as a reliable negative acknowledgement. Therefore, in each round, Alice incurs a cost of roughly $\sqrt{\ell}$ in expectation while the adversary incurs a cost of 1. Through this attack, the adversary can force Alice to quickly deplete her energy supply.

**ROBUST COMMUNICATION:** A better resource-competitive algorithm from [43], ROBUST COMMUNICATION, is given in Figure 1. While similar to LESS NAIVE, there are important differences.
The length of the Send Phase increases geometrically with the round index $i$ and a “mystery” constant $c > 0$ that we discuss later. Alice and Bob still act probabilistically, but their probabilities differ.

In the Nack Phase, the number of slots also increases geometrically with $i$, and Bob is required to send nack in all slots in order to prevent Alice from terminating. In this way, Bob makes a “down payment” before proceeding to the next round. In the Nack Phase, Alice samples an expected $O(1)$ slots and, if all such slots either contain nack or are blocked, she continues into the next round. An adversary who spoofs Bob may attempt to save energy by sending in only some of the $2^i$ slots; however, Alice is likely to detect this trickery and terminate.

We must have $c > 1$; otherwise, Bob’s listening probability in the Send Phase is nonsensical. Conversely, we must have $c < 2$; otherwise, the adversary can drain Alice’s energy by spoofing Bob. The constant $c$ controls the cost functions between Alice, Bob, and the adversary. If we desire a fair protocol, one where Alice and Bob spend roughly the same amount relative to the adversary, we can optimize for $c$. The following result was proved previously:

**Theorem 2.** (King, Saia, and Young [43]) Assume an adversary that jams an unknown number $T$ slots. ROBUST COMMUNICATION guarantees communication with an expected cost to Alice and Bob of $O(T^{1/\phi} + 1) = O(T^{0.62} + 1)$ where $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

Using Definition 1, the cost function $\rho(T) = O(T^{1/\phi}) = O(T^{0.62})$ and the efficiency function $\tau = O(1)$ in expectation. Therefore, when there is no jamming, the players succeed quickly and with small cost. Alternatively, if there is significant jamming, the faulty devices will deplete their aggregate energy budget rapidly and then the players will succeed. In this latter case, the adversary is effectively bankrupted! For the energy-constrained domain of wireless sensor networks, Theorem 2 provides a convincing proof-of-concept of a resource-competitive algorithm.

### 2.1 Defining Costs and Resources

In our proof-of-concept, the costs for accessing the wireless channel were assumed to be a normalized cost of 1. However, in practice, such costs would be measured in milliwatts; for example, sending (at 0 dBm) and listening dominate the operating costs of the popular Telos mote at 35mW and 38mW, respectively [59]. While this is a reasonable match to the Alice and Bob scenario, how do we reason about costs in general?

To answer this question, we make two points. First, while the unit costs in the Alice and Bob scenario represent an abstraction, if the actual costs for sending, listening, and jamming are approximately the same order of magnitude, our asymptotic results should correspond to reality. More broadly, when considering an assignment of costs, we need not be concerned with absolute numbers so long as the magnitude of the costs to the good nodes relative to the bad nodes is correct.

Second, while the decision about what constitutes a “resource” should be well motivated, this should not be a difficult task for many technological domains. For example, the past two decades, and the literature on the future of wireless devices [4, 17, 69], implies that energy will continue to be scarce in these networks. In general, CPU cycles, bandwidth, and energy have been viewed as “resources” since the inception of modern-day computing and this seems unlikely to change.
3 Related Work

We compare and contrast resource competitiveness with a number of related concepts.

Notions of Relative Cost. Competitive analysis is a well-known technique where one evaluates the worst-case performance of an online algorithm relative to an optimal offline algorithm $OPT$ [68]. While the inputs to an online algorithm can be viewed as adversarially selected, there is no notion of cost to the adversary for selecting certain inputs over others. In contrast, resource competitiveness places the cost to the adversary directly in the performance metric (see Definition 1); this is a key difference. Unlike online analysis where it is impossible for an online algorithm to outperform $OPT$, a resource-competitive algorithm can actually be more efficient than an adversary that tries to attack.

Parameterized analysis is a general technique that has been used in many contexts such as online paging algorithms [26, 53], graph theory [10], and the traveling salesperson problem [7, 9]. Resource competitiveness might be viewed as extending this approach to the distributed setting, where it is often natural to consider an actual struggle with an adversary, and where a careful modeling of costs is necessary.

Game theory provides another measure of competitiveness known as the “price of anarchy” which is the ratio of the worst-case Nash equilibrium to the global social optimum [54, 64]. In resource-competitive analysis, each node either obeys the protocol or it does not; in game theory, nodes seek to maximize their respective utility functions. It is possible to address malicious behavior in the context of game theory (see [2, 3, 19, 48, 72]). The incorporation of game theoretic concepts may be an interesting direction for future work on resource-competitive algorithms; however, the current challenges in designing resource-competitive algorithms are sufficiently formidable to warrant their investigation first.

Notions of Inflicting Cost. The idea of inflicting cost on an opponent arises more explicitly in the domain of cryptography. In [23], Diffie and Hellman state that the goal in designing a cryptographic system “is to make the enciphering and deciphering operations inexpensive, but to ensure that any successful cryptanalytic operation is too complex to be economical.” This idea that an attacker is burdened by a disproportionate cost in attempting to break a cryptosystem underlies all modern-day cryptosystems.

A major differentiating aspect of cryptographic approaches is that a security parameter, e.g. a length of a private key, is decided prior to running the algorithm. This roughly determines (i) how much the adversary must spend in order to compromise the cryptosystem, and (ii) how much the good nodes must spend to achieve a particular level of security. In contrast, resource-competitive algorithms are adaptive, as described in Section 1.1. Recall that when $T = 0$, there is a small upfront cost quantified by the efficiency function $\tau$. Then, as $T$ increases, the cost function $\rho(T)$ increases in response and will dominate the cost of the algorithm when $T$ grows large enough. This is very different from having a predetermined cost.

Additional Related Results. An early example of considering an attacker’s resources involves a public-key cryptosystem by Merkle [51] where computational puzzles are used to inflict cost on an eavesdropper. However, the hardness of the puzzles must be set a priori and, therefore lacks the adaptivity of resource-competitive algorithms.
Inflicting computational cost has been used to deter spam email [27]. Another example arises in settings where an attacker controls multiple identities. In such a network, one may issue a cost for joining via computational puzzles [47, 70] or monetary penalties [15]. Even the social cost of establishing links between two nodes in a social network graph has been exploited to limit the impact of such attacks [77]. Similar ideas of inflicting cost have been used to mitigate application-level DDoS attacks. Typically, the focus is on requiring a client to spend bandwidth or computational resources prior to receiving service (see [49, 56, 73]).

Assuming that the adversary does not have substantially more resources than the good nodes, these approaches impose a cost that limits the adversary’s ability to disrupt the system. In contrast, rather than make this assumption, a resource-competitive algorithm \( A \) quantifies a cost relationship between the adversary and the good nodes. This property of the algorithm allows us to better understand the performance of the system under any amount of disruption. For example, one may show that the adversary requires more resources (and how much more) in order to thwart \( A \) (as exemplified in Section 2). Conversely, if good nodes are at a disadvantage, then the magnitude of this disadvantage is known and can be used when provisioning the system in order to still give guarantees.

4 Recent Resource-Competitive Results

Recently, a number of resource-competitive results have appeared:

**Large Wireless Sensor Networks:** Following up on [43], the problem of tolerating jamming attacks in larger wireless ad-hoc networks has been investigated with similar advantages to the good nodes in Gilbert and Young [38] and Gilbert et al. [36]. Together, these three works demonstrate that there is hope for dealing with adversaries that can employ more general jamming strategies.

**Robust Backoff:** Randomized binary exponential backoff (or simply backoff) is a well-known and widely deployed technique for coordinating access to a shared resource, such as a communication channel (see [52] for wired networks, and [44, 75] for wireless networks). Several works examine the performance of (variants of) backoff under various models of process arrivals (for examples, see [8, 18, 33, 39–41, 74]). However, prior results cannot guarantee constant throughput in the fully dynamic case where the number of processes seeking access to the channel can change arbitrarily from one time step to the next.

In addition to throughput, another important metric is the number of access attempts performed by a process prior to successfully utilizing the channel. This number should be relatively small; otherwise, bandwidth is wasted and, in the case of an excessive number of attempts, throughput is degraded. Recall that interference on the communication medium may render the channel unavailable. If disruption leads to \( T \) time slots of unavailability, then the number of access attempts should be a small function of \( T \).

A recent result by Bender et al. [6] addresses these challenges. The authors demonstrate a resource-competitive algorithm that yields constant throughput and where each process makes only an expected \( O(\log^{O(1)} T + \log^{O(1)} n) \) access attempts, where \( n \) is the total (unknown) number of process arrivals over the lifetime of the system.
**Interactive Communication:** Alice and Bob wish to communicate over a noisy binary channel. The goal in interactive communication is to give an algorithm that takes as input some distributed protocol $\pi$ that executes over a noise-free channel, and outputs a distributed protocol $\pi'$ that works over the noisy channel. Communication over the channel is synchronous, and a *channel step* is defined to be the amount of time taken to send one bit over the channel. Given that $\pi$ works over $L$ channel steps, the goal is to minimize the number of channel steps required by $\pi'$ and, therefore, reduce the bandwidth used.

A number of important results have been established [11–13, 31, 32, 34, 55, 66, 67]. However, common to all of these previous works is a focus on tolerating the maximum possible noise rate, but how do these algorithms perform with smaller noise rates? This question was the subject of recent work by Haeupler [42] who noted that these schemes incur a large communication overhead for smaller noise rates. Motivated by this inefficiency, Haeupler [42] demonstrated an algorithm that achieves a conjectured near-optimal overhead for a given noise rate which is *known* in advance.

What if the noise rate is unknown? In this case, for an arbitrary and unknown $T \geq 0$ bit flips, work by Dani et al. [22] provides a resource-competitive algorithm that succeeds with high probability in $L$ and, if successful, has an expected cost of $L + O((T + \sqrt{LT} + L) \log(LT + L))$ channel steps.

**Reliable Bridge Distribution in Tor:** Tor is the largest anonymous communication network that allows users to access the Internet anonymously by providing them with the option to connect to a set of servers called bridges. In networks that are under censorship, adversarial authorities can inject dishonest users in the network in order to learn the addresses of bridges and then block them. A challenging problem is to distribute a set of bridges among $n$ users in such a way that all honest users are guaranteed to be able to connect to Tor in the presence of an adversary corrupting $T < n$ number of users. In this setting, the power of the adversary, and thus $T$, is usually unknown.

Crandall et al. [20] describe a randomized bridge distribution algorithm when $T$ is unknown. Their algorithm is resource-competitive; it adaptively increases the number of bridges according to the behavior of the adversary. It requires $\tilde{O}(T)$ bridges and the number of times a user fails to connect to Tor via bridges is bounded by $O(\log T)$ with high probability.

While each of these results parameterize algorithmic performance by $T$, the implications of the cost relationship differ. For wireless sensor networks, our concerns are security oriented: we attempt to thwart a denial-of-service attack by bankrupting the attacker. In constrast, for backoff and interactive communication, we are concerned with (bandwidth) efficiency and we seek to minimize the communication overhead relative to the amount of disruption. This diversity of results illustrates how resource competitiveness can be applied to achieve results in a variety of settings.

## 5 Conclusion and Future Work

Resource competitiveness is a useful addition to the collection of tools that algorithmicists can use for designing fault-tolerant systems. In security settings where both good and bad devices are resource constrained, this technique allows us to compete with an attacker. Under more benign fault models, efficient robustness to disruption can be obtained.
There are a number of open problems that seem amenable to a resource-competitive approach. To date, results addressing jamming in wireless sensor networks have been confined to single-hop networks, and it would be interesting to see a result for a multi-hop setting. Additionally, the analogue to jamming attacks in the wired domain are denial-of-service attacks, and it would be of interest to determine what results are possible under such attacks.

While [36, 38] addresses the challenge of broadcast, this does not lead directly to a resource-competitive algorithm for leader election, consensus, or Byzantine agreement. Results pertaining to these canonical distributed computing problems would be of interest.

Acknowledgements. We are grateful to Valerie King for her valuable suggestions in writing this article.

References


