# An Extremum Seeking Algorithm for Message Batching in Total Order Protocols

Diego Didona
*IST/INESC-ID, Lisbon, Portugal*

Daniele Carnevale, Sergio Galeani
*Rome University Tor Vergata, Italy*

Paolo Romano
*IST/INESC-ID, Lisbon, Portugal*

***Abstract*—Message batching is a well-known optimization technique to maximize throughput of networked services. The manual configuration of the appropriate batching level is however a time consuming and not trivial task. Too low batching values can in fact render the system unstable in presence of high loads; excessively high batching values, on the other hand, can lead to high latency at low load, which may be unacceptable for delay sensitive applications. The problem is further exacerbated in presence of fluctuating workloads,as in these scenarios the optimal batching level varies dynamically over time, and pursuing optimal performances demands the employment of self-adaptive mechanisms.**

**In this paper we study the problem of self-tuning the message batching level adopting an interdisciplinary approach that employs methodologies from control theory community to optimize the performance of Total Order Broadcast (TOB), a fundamental building block to build dependable distributed systems.**

**Specifically, we introduce an innovative self-tuning algorithm based on extremum seeking optimization principles. We provide theoretical results on its convergence properties and an extensive experimental analysis aimed at assessing the actual effectiveness of the new algorithm in a state-of-the-art group communication system.**

## I. Introduction

Message batching [1] (also known as message packing [2] or message aggregation [3]) is a simple, yet very effective, optimization technique that is based on the idea of buffering messages for some time, so to be able to process multiple messages together. This allows amortizing the costs of processing each individual message in the batch, reducing the header overhead per message, the contention on the network and the CPU load [4]. Indeed, several works have highlighted the striking impact of batching in boosting system throughput. On the other hand, at low load, waiting for additional messages to form a batch induces unnecessary stalls that can hamper the performance of delay sensitive applications, e.g., interactive, real-time applications.

The problem is exacerbated in the presence of dynamic, fluctuating workloads. In this (in practice quite common) case, the optimal batching factor actually varies over time, making any static configuration policy clearly suboptimal.

In this paper we address the problem of designing and implementing a self-adaptive scheme for dynamically tuning the message batching level employing techniques and mathematical analysis methodologies originated in the control theory community. The proposed self-tuning mechanism is based on the *extremum seeking* method, a model-free control technique that aims at optimizing the steady-state relationship between the system input parameters and its performance output. Being a model-free approach, this relationship is assumed unknown, and the optimum is searched relying exclusively on the monitoring of the system under control. In an iterative fashion, this control technique perturbs the input parameters, observes the response, and tunes them accordingly.

Our self-tuning algorithm, which we call Adaptive Extremum Seeking Controller (AESC), uses multiple instances of extremum seeking optimizers: each instance is associated with a distinct value of batching $b$ and learns the corresponding optimal waiting time for a batch of size $b$. The simultaneous usage of multiple instances of extremum seeking optimizers ensures high responsiveness to quick variations of the message arrival rate. Further, the exploratory nature of the extremum seeking technique makes the controller robust to external perturbations, possibly of transient nature, which may induce dynamic shifts of the speed/efficiency of the machine hosting the sequencing process (due, e.g., to the activation of additional software processes). As a further optimization, AESC employs an adaptive exploratory step, computed as decreasing function of the batch size, which allows minimizing both overshooting phenomena and the amplitude of oscillations around the optimal value.

We provide a rigorous analytical characterization of the convergence properties of the proposed self-tuning algorithm, which allows us to outline the assumptions necessary to ensure both the optimality of the solution found as well as its stability. Our theoretical analysis considers first an ideal controller, capable of enforcing the control law with perfect timing accuracy. As software-based implementations of such an ideal controller are infeasible on off-the-shelf platforms, we extend our analysis to consider a more realistic controller model, whose logic is triggered asynchronously, i.e., upon the reception of a new message, or synchronously, but with a coarse grain periodicity.

We apply the proposed self-tuning technique to optimize the performance of a Total Order Broadcast (TOB) algorithm [5]. TOB represents a fundamental problem in distributed systems, which requires a group of distributed processes to reach agreement on a common order of delivery of messages, in presence of concurrent broadcasts by any process of the group. The relevance of TOB stems from the fact that it

represents an essential building block for a wide range of distributed coordination problems, and has been employed in a wide range of heterogeneous application domains [6]–[9]. In this paper we focus on Sequencer-based TOB (STOB) algorithms [10]. This is a class of algorithms that is particularly popular as it guarantees theoretically optimal latency [11]. On the other hand, its maximum throughput is upper bounded by the capacity of the sequencer to generate sequencing messages. Interestingly, the sequencer capacity can be greatly increased by using an adequate batching level. Hence, STOB represents an ideal candidate for assessing the effectiveness of the proposed mechanism for self-tuning the batching level.

We integrated AESC in the Appia Group Communication System [12], and performed an extensive experimental evaluation aimed at assessing its performance both in terms of responsiveness (overshooting and convergence speed) to input signal variations, as well as steady state optimality.

The rest of the paper is structured as follows. Section II discusses related works. In Section III and Section IV, we provide an overview of the (S)TOB algorithms and extremum seeking control technique, respectively. The AESC algorithm is presented in Section V. Its convergence analysis is introduced in Section VI, whereas Section VII presents the result of our experimental study. Section VIII concludes the paper.

## II. Related Work

Message batching is a well known optimization that is commonly employed in several domains [1], [3], [4]. TCP Nagle's algorithm [13] is a notorious instance of such a technique.

The effects of batching on the performance of TOB protocols was first studied empirically in [4] and later mathematically in [3].

The work in [1] was one of the first ones to define a lightweight, not intrusive architecture for message batching in TOB protocols. However, the techniques proposed in this work require the explicit setting of additional parameters, e.g. the duration of timers used to wait for messages to be batched. Hence, they do not fully automatize the tuning of the batching mechanism.

The only self-tuning solutions for message batching that we are aware of are those presented in [2], [14]. The first one can be seen as an adaptive optimization discrete time algorithm that performs on-line searches to retrieve the optimal batching $b^\star(t)$ changing periodically the batching levels of one unit. However, the approach that we propose here is quite different for several reasons. First, AESC does not change (probing) directly the batching level. It probes the response of the system while varying the waiting time $T_b$ associated with a given level $b$ of batch, in order to identify the optimal time to wait, since the processing of the last batch, before processing the next batch of $b$ messages. By employing multiple, extremum seeking optimizers associated with different batch levels, AESC maintains (although in an indirect way) memory of the previously identified ideal level of batching for a given arrival rate. Further, AESC uses an adaptive probing step, combined with a dithering signal, rather than a static adaptation step. As it will be also confirmed by our evaluation study, these design differences allow AESC to achieve significantly higher performance, in terms of robustness to noisy measurements, convergence speed and stability.

The solution in [14] presents a reinforcement learning based optimization technique that is bootstrapped by an analytical model that exploits queuing theory arguments. Reinforcement learning is used to ensure eventual convergence to the optimal batching configuration, correcting possible initial errors of the analytical model. The analytical model allows increasing the convergence speed of the reinforcement learning, reducing the chances of exploring very likely suboptimal batching configurations. AESC differs from this solution for two fundamental aspects. It uses a model free approach, which spares from the need of conducting off line benchmarking experiments to identify the system-dependent parameters necessary to instantiate the analytical model. Also, the solution in [14] requires to gather stable statistical average on the message arrival rate, which requires observation windows of the order of seconds, limiting the responsiveness of the controller. By discretizing with respect to the batching level, instead than on the message arrival rate, AESC avoids *a priori* this issue.

Our work is also related to performance evaluation and modelling studies of TOB [15], [16] (and related agreement problems, consensus *in primis* [17], [18]).

Finally, our work has also relations with the ones exploring the usage of machine learning techniques to forecast the performance of computer systems in several contexts. These include solutions aiming at forecasting the throughput of TCP flows [19], Pub-Sub systems [20], and Atomic Broadcast protocols [21], at automatically classifying traffic based on semi-supervised learning techniques [22], at automatizing the allocation of resources in cloud-computing infrastructures [23], and at generating software ageing models [24].

## III. Overview of the STOB Algorithm

As already discussed in the Introduction section, STOB algorithms are probably among the most widely deployed TOB protocols [12], as they achieve the minimum bound on message latency for the TOB problem. Different variants of STOB protocols have been compared in the literature, e.g., with fixed and dynamic leader [4] or with and without uniform delivery guarantees [25]. In this work we focus on the simplest of the STOB algorithms, namely a STOB algorithm which does not guarantee message uniformity, and in which the sequencer role is statically assigned (unless in presence of group membership changes). This choice is

made essentially for the sake of simplicity. Nevertheless, all of the aforementioned variants can exploit the batching optimization without additional difficulties, and the self-tuning techniques described in this paper could be adapted to be integrated in more complex variants of this family of TOB algorithms. In the remainder of the paper we shall refer, for simplicity, to the nonuniform, static STOB algorithm described in the following simply as to STOB.

In failure-free runs of the STOB algorithm, if no processes leave or join the group, the processes agree on the identity of a single process, before starting to totally order broadcast (TO-Bcast) messages. Such a process, called sequencer, has the role to impose a common total order of delivery on messages to all processes in the group. If a process wants to totally order broadcast a message, it executes a plain broadcast of the message. When a process receives a message from the network, however, it cannot immediately totally order deliver (TO-deliver) it to the application. In order to guarantee group-wide agreement on the final delivery order, in fact, it has first to wait to receive from the sequencer the corresponding *sequencing* message, and to ensure that all previously ordered messages have been delivered.

The batching level, denoted in the remainder as $b$, defines how many messages the sequencer waits to receive before generating a sequencing message. As already discussed, setting $b$ to 1 ensures minimal latency at low load. At high loads, however, higher values of batching allow to amortize the cost of sequencing each message, and the sequencer to sustain much higher throughput rates.

## IV. Overview of Extremum Seeking Controllers

*Extremum seeking* controllers were introduced in the early '50s (see [26] and [27]), and have been largely studied by the control community as a mechanism to synthesize adaptive controllers for maximizing/minimizing certain system performance indexes. Nevertheless, a formal proof of the classical extremum seeking scheme has been given only recently in [28]. This work proved local stability properties of an extremum seeking feedback scheme for general nonlinear systems and motivated further interesting researches and applications [29], [30], [31], [32], [33]. More recently, a nonlocal result has been proposed in [34] and a unifying approach using adaptive schemes based on the estimation of output derivatives are given in [35], that together with [36] and [37], link the classical extremum seeking algorithms with the tools for static optimization [38] such as gradient-based [39], Newton [39] and line-search [40] algorithms.

In the classical extremum seeking algorithm the input that improves a cost/performance index $J$, which is a function of the state of the system, is mainly obtained by evaluating the approximate directions (descending/ascending) that improve the index $J$ exploiting a two time scale separation [28] among the system (also called "plant") and controller dynamics. In general, the controller is set to be "slower" than the controlled system, such that the plant output, on which the index $J$ is evaluated, can be considered to be in "steady-state" with respect to the slowly changing input of the extremum controller. This allows to have a measure of the index $J$ that essentially is not affected by the transients induced by the plant dynamics, which may potentially ruin the approximate evaluation of the descent/maximizing directions. Such directions are then exploited by the controller to define the correct input to the plant.

A key ingredient of the extremum seeking scheme is the *dithering signal*, which is a time varying signal that probes different directions in the input space of the plant, providing an approximate gradient of $J$. Common dithering signals are sinusoids and square waves.

As in [33], the dithering signal can be substituted by noise acting on the system that continuously changes the plant input. As will be discussed next, we do exploit such noises in the technique we propose to optimize the plant latency through the "best" selection of the batching values.

```
long T = 0          // current (at iteration k) batch waiting time
perfIndex J = 0     // value of the target performance index
                    // at previous iteration (at iteration k − 1)
int δ = 1           // current probing direction (1=inc., -1=dec.)

UPDATE(long new_J_value)
  // confirm δ if the current T yielded a better (lower) J value
  // otherwise, invert δ
  d = generateNewDitheringValue()
  δ = sign(δ) · sign(J − new_J_value)
  γ = max{γ_min, γ_0/b}
  T = T + δ · γ + d
  J = new_J_value
```

Figure 1. Pseudocode for the Extremum Seeking Optimizer (ESO) associated with batch size $b$

## V. The AESC algorithm

Fig. 1 and Fig. 2 report the pseudocode describing the AESC self-tuning algorithm, assuming, to simplify presentation, an ideal, event-based controller which is activated instantaneously whenever the conditions of its triggering event become valid. In Section V-A we shall discuss how to adapt the logic of the AESC algorithm to allow its efficient implementation using a purely software-based approach.

As already mentioned, AESC relies on a series of independent extremum seekers optimizers, denoted as ESOs in the following, each one associated with a different batching level $b$, where $b \in \mathcal{B} \triangleq \{1, \ldots, b_{\max}\}$. The behavior of each ESO is defined by the pseudocode in Fig. 1.

The state of an ESO associated with batching level $b$ is composed of the three following pieces of information: the current (say at iteration $k$) waiting time, denoted as $T$, for a batch of size $b$; the value $J$ of the target performance indicator at the previous iteration (namely at iteration $k-1$);

$\delta \in \{-1, 1\}$, namely the current direction (increasing vs decreasing) of the exploratory step.

The ESO's state is updated by using the simple logic defined by the UPDATE() method. This method takes as input parameter the value (measured from the system) of the performance index obtained using the current batch waiting time $T$, and confirms/inverts the seeking direction based on whether the just measured performance index (at iteration $k$) is better than the one that had been measured at the previous iteration (at step $k-1$). The update rule includes also a dithering signal, over whose generation we abstract with the `generateNewDitheringValue()` primitive. For example, the dithering can be a random noise, a sinusoid or a square wave signal of "small" amplitude that allows continuous probing of the descending directions.

The width of the exploratory step used to update the value of $T$ at iteration $k+1$, denoted as $\gamma$, is computed as the maximum between $\frac{\gamma_0}{b}$ and $\gamma_{min}$, where $\gamma_0$ and $\gamma_{min}$ are two algorithm parameters that allow to tune the reactivity of the optimizers and whose recommended settings will be discussed in the following. This update rule is aimed at reducing the amplitude of the exploratory step for large batching values, and is motivated by the following rationale: large batching values result optimal (and hence likely to be used by the AESC algorithm) at high load, i.e., in presence of high message arrival rates; narrowing the probing step associated with large batching values allows therefore achieving a finer grained granularity precisely in load scenarios where small variations of the batch waiting time have a high impact on the probability of receiving additional messages. As we will see in Section VII, this optimization allows to enhance significantly the dynamic performances of our controller, reducing both overshoots and oscillations. Further, the update rule of $\gamma$ guarantees that $\gamma \in [\gamma_{min}, \frac{\gamma_0}{b}]$. Based on our experimental study, in fact, we noticed that enforcing a lower bound on $\gamma$ was necessary in order to prevent the optimizer from getting stuck in local minima due to the choice of excessively small exploratory step in presence of (unavoidably) noisy measurements.

The main control logic of the AESC algorithm is defined by the pseudocode of Fig. 2. It assumes an ideal controller whose logic is triggered whenever the time elapsed since the delivery of the last batch grows larger than the batch waiting time determined by the ESO instance associated with a batch value equal to the number of messages batched so far (and stored in a message buffer denoted as $msgQ$). In this case, the current batch of messages is delivered, using the `deliver()` primitive, which we assume to return also the performance index measured upon the delivery of the batch.

AESC adopts the self-delivery latency measured at the sequencer side as performance index to evaluate the goodness of the current setting of the waiting time for batch size $b$ (namely, $ES_b.T$). Note that in principle one may

```
array of ESO ES[1 ... b_max]
Set msgQ = ∅                    // buffer for batching msgs
timestamp lastDelTime = 0  // last batch delivery time

upon msgQ.size() > 0 ∧
        ( now() − lastDelTime > ES_msgQ.size().T )
 int b=msgQ.size()
 perfIndex J =deliver(msgQ)
 ES_b.UPDATE(J)
 ENFORCEMONOTONICITY(b)
 lastDelTime=now()

ENFORCEMONOTONICITY(int b)
 if (ES_b.J > 0)
   foreach(b' > b)
     ES_b' = max{ES_b'.T, ES_b.T}
 else
   foreach(b' < b)
     ES_b' = min{ES_b'.T, ES_b.T}
```

Figure 2. Pseudocode of the Adaptive Extremum Seeker Controller

use alternative performance indexes, but the self-delivery latency has the appealing property of being measurable instantaneously, differently, for instance, from throughput (as in [2]), which requires an observation over an appropriately sized time window.

The measured value of the performance index is then used to update the state of the corresponding ESO. Finally, an additional check is performed to enforce the monotonicity, and specifically the not decreasing nature, of the function that correlates the batch size and the batch waiting time (which we denote as $T_b$ in the following). This is necessary to guarantee the meaningfulness of the batch waiting time information stored across the various instances of $T_b$ and to enforce the obvious correctness conditions that, given two batch sizes $b$ and $b'$, with $b' > b$, the waiting time for a batch of size $b'$ has to be larger than that for a batch of size $b'$, i.e., $T_{b'} > T_b$.

### A. Non-ideal controller

The above described controller assumes that the condition triggering the delivery of a batch can be evaluated instantaneously, namely as soon as it becomes verified.

In practice, however, the evaluation of the batch delivery condition has to be implemented using a periodic timer. In commodity, software-based implementations (e.g., in the JAVA-based Group Communication Toolkit used in our evaluation study [12]), the period granularity is typically on the order of at least several milliseconds, due both to accuracy issues and to the overhead imposed by mechanisms that do support high frequency samplings (such as spin-locking).

Hence, a more realistic AESC implementation (such as the one we developed) can evaluate the batch delivery condition only when a new message arrives, or when a certain amount

of time $\overline{T}$ (normally on the order of a few milliseconds) elapses since the last batch has been sequenced.

This implies that the batches are closed with a delay $e_b(k) \geq 0$ with respect to the selected waiting time $T_b(k)$. Similarly to [33], where a noise acting on the plant is used as a dithering signal, also AESC exploits the delay $e_b(k)$ to generate the dithering, which our experiments have revealed to be sufficiently small not to compromise the probing of the descending direction of $J$.

## VI. Convergence analysis

In this section we analyze the convergence properties of the AESC algorithm. We start by precisely identifying the necessary assumptions for AESC convergence (whose validity in a system will be confirmed in our experimental study of Section VII). Next we demonstrate the global asymptotic stability of AESC, i.e. that each individual instance $\text{ES}_b$ of the extremum seeking optimizers identifies, after a finite time, a batch waiting time $ES_b.T$ that is within an estimable distance from the optimal waiting time for batch size $b$, namely $ES_b.T^*$.

In order to simplify the notation, in the following we will use $T_b$ to refer to $ES_b.T$, and $T_b(k)$ to denote the value of $T_b$ after the $k$-th update of $ES_b$. Further, we will use the notation $J(b,T)$ to refer to the value of the performance index (i.e., the self-delivery latency as measured at the sequencer side) perceived when adopting a waiting time $T_b$ for a batch size $b$.

Note that, since we use the self-delivery latency as our performance index, we need to prove that AESC correctly minimizes the function $J(b,T)$ for any value of $b \in \mathcal{B}$

We can now introduce the two necessary assumptions that are instrumental to prove the convergence of AESC.

*Assumption 1:* There exists a class-$\mathcal{K}$ function[1] $\alpha_l(\cdot)$ and a unique $T_b^\star$ such that

$$J(b, T_b^\star + \delta) - J(b, T_b^\star) \geq |\delta| \alpha_l(|\delta|), \tag{1}$$

for all $\delta \in \mathbb{R}$ and $b \in \mathcal{B}$. □

Inequality (1) implies that $J(b,\cdot)$ is in the incremental sector[2] $(0,\infty)$ around $T_b^\star$, yielding also that $T_b^\star$ minimizes the cost function $J(b,\cdot) : \mathcal{B} \times \mathbb{R}_{>0} \to \mathbb{R}_{>0}$, i.e.

$$T_b^\star \triangleq \arg\min_{T \in \mathbb{R}_{>0}} J(b,T). \tag{2}$$

*Assumption 2:* The dithering signal $d_b(k)$ is such that $0 < |d_b(t)| \leq d_0$ for all $k \geq 0$ and $b \in \mathcal{B}$. □

In the next proposition we prove that the algorithm described in the previous section, implemented using an ideal controller and a fixed probing step of amplitude $\gamma_0$, allows to continuously *minimize* the cost function $J(b, T_b(k))$ via the update of $T_b(k)$ and the dithering signal $d_b(k)$. We relax the assumption of ideal controller and static gain $\gamma(k) = \gamma_0$ in the remainder of this section.

*Proposition 1: (ideal case)* Let Assumption 1 and Assumption 2 hold and select the update law of the waiting time as

$$T_b(k+1) = T_b(k) - \gamma_0 \operatorname{sign}\left(J(b,T_b(k)) - J(b,T_b(k-1))\right) \times \operatorname{sign}\left(T_b(k) - T_b(k-1)\right) + d_b(k), \tag{3}$$

with $d_0 < \gamma_0$ and set

$$T_j = \max_{j=b+1..b_{\max}} \left(T_b, T_j\right), \text{and } T_j = \min_{j=1..b-1} \left(T_b, T_j\right), \tag{4}$$

then the set $\mathcal{A}_b \triangleq [T_b^\star - 2\gamma_0 - d_0,\, T_b^\star + 2\gamma_0 + d_0]$ is globally asymptotically stable and there exists $\kappa(T(0))$ such that $T_b(k) \in \mathcal{A}_b$ for all $k \geq \kappa(T(0))$ and any initial waiting time $T_b(0)$.

*Proof:* Since

$$\operatorname{sign}(J(b,T_b(k)) - J(b,T_b(k-1))) \operatorname{sign}(T_b(k) - T_b(k-1))$$

is positive when an increment of the $T_b$ produced an increment of the cost function $J$ or, equivalently, when a decrement of $T_b$ induced a decrement of $J$, then by Assumption 1 the waiting time at the next step is decreased to minimize $J$. On the other way round, when a decrement of the $T_b$ produced an increment of the cost function $J$ or, equivalently, when an increment of $T_b$ induced a decrement of $J$, then by Assumption 1 the waiting time at the next step is increased to minimize $J$. The asymptotic stability proof of the set $\mathcal{A}_b$ directly follows from the above considerations, Assumption 1, and $d_0 < \gamma_0$ yields $V(T_b) = J(b,T_b) - J(b,T_b^\star)$ to be a positive definite Lyapunov function [41] such that

$$V(T_b(k+1)) - V(T_b(k)) \leq -(2\gamma_0 + d_0)\alpha(\operatorname{dist}_{\mathcal{A}_b}(T_b(k))) \tag{5}$$

holds for any $T_b(k) \notin \mathcal{A}_b$, where $\operatorname{dist}_{\mathcal{A}_b}(T_b(k))$ is the distance of $T_b(k)$ from the set $\mathcal{A}_b$ defined as $\operatorname{dist}_{\mathcal{A}_b}(T) \triangleq \inf_{s \in \mathcal{A}_b} |s - T|$. Then, since $V$ is positive definite and the right-hand side of (5) is negative as long as $\operatorname{dist}_{\mathcal{A}_b}(T_b(k)) > 0$, the $\operatorname{dist}_{\mathcal{A}_b}(T_b(k))$ has to go to zero as $k$ grows to infinity, i.e.

$$\lim_{k \to \infty} \alpha(\operatorname{dist}_{\mathcal{A}_b}(T_b(k))) = 0.$$

Furthermore, defining $\delta_0 > 0$ such that $\gamma_0 = \delta_0 + d_0$, then for any $T_b(0) \notin \mathcal{A}_b$ the maximum number of steps $\overline{k}$ such that $T_b(k) \in \mathcal{A}_b$ for all $k \geq \overline{k}$ is given by

$$\overline{k} = \frac{\operatorname{dist}_{\mathrm{A}_b}(T_b(0))}{\delta_0}.$$

■

*Remark 1:* Note that the procedure described by (4) guar-

[1] A class-$\mathcal{K}$ function $\alpha(\cdot) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is continuous, monotonically increasing and such that $\alpha(0) = 0$. See [41] for further details.

[2] A function $f : \mathbb{R} \to \mathbb{R}$ is in the incremental sector $(0,\infty)$ if $\nabla f$ is in the sector $(0,\infty)$.

antees better transients based on the monotonicity condition $T^\star_{b+1} \geq T^\star_b$ in Assumption 1 and does not affect stability of the overall procedure.

*Remark 2:* The inequality $\gamma_0 > d_0$ is not a necessary but a sufficient condition that renders easier the proof of Proposition 1. If $d_0 \geq \gamma_0$, then we need further assumptions on $d_b(k)$, such as zero mean value, and this requires more involved mathematical tools for stability/attractivity analysis. □

Let us now consider the case of nonideal controller in which the closure of a batch size of size $b$ can be affected by a generic delay $e_b(k)$. The next proposition provides stability conditions under which such a nonideal controller yields global asymptotic stability of a new set $\mathcal{A}_b$.

*Proposition 2:* Define $\delta_{e,b}(k) = e_b(k) - e_b(k-1)$ as the difference of two consecutive delays affecting the closure of the batch of size $b$, and $\delta_{T,b}(k) = T_b(k) - T_b(k-1)$ as the difference of two consecutive waiting times selected for the batch of size $b$. Let Assumption 1 and Assumption 2 hold and select the update law of the waiting time as in (3) with $d_b(k) \equiv 0$; then if $|\delta_{T,b}(k)| > \delta_0 \geq |\delta_{e,b}(k)|$ holds for all $k \geq 0$ and some $\delta_0$, the set $\mathcal{A}_b \triangleq [T^\star_b - 2\gamma_0 - \delta_0,\, T^\star_b + 2\gamma_0]$ is globally asymptotically and there exists $\kappa(T(0))$ such that $T_b(k) \in \mathcal{A}_b$ for all $k \geq \kappa(T(0))$ and any initial waiting time $T_b(0)$.

*Proof:* The delays renders (3) as

$$T_b(k+1) = T_b(k) - \gamma_0 \operatorname{sign}\Big(J(b, T_b(k) + e_b(k)) - J(b, T_b(k-1) + e_b(k-1))\Big)\operatorname{sign}\Big(T_b(k) - T_b(k-1)\Big), \tag{6}$$

which is still useful to move along the minimizing directions of $J$ if

$$\operatorname{sign}\Big(T_b(k) + e_b(k) - (T_b(k-1) + e_b(k-1))\Big) = \operatorname{sign}\Big(T_b(k) - T_b(k-1)\Big), \tag{7}$$

and, by $|\delta_{T,b}(k)| > \delta_0 \geq |\delta_{e,b}(k)|$, it holds

$$\operatorname{sign}(\delta_{T,b}(k)) = \operatorname{sign}(\delta_{T,b}(k) + \delta_{e,b}(k)). \tag{8}$$

To conclude, arguments similar to the ones of the proof of Proposition 1 can be used to state the global asymptotic stability of the set $\mathcal{A}_b$ and the existence of $\kappa(T(0))$. ■

The condition $|\delta_{e,b}(k) < \delta_0|$, defining with $m(k)$ the arrival rate of messages, can be rewritten as $|1/m(k) - 1/m(k-1)| < \delta_0$. This implies that if the arrival rate of the messages changes sufficiently slowly the results of Proposition 2 holds. It will be shown by the experimental tests that the algorithm of Proposition 1 is very robust even if such instantaneous condition on the derivative of the inter-arrival rate is not satisfied (abrupt changes of message arrival rate is forced).

Note that the sets of Proposition 1 and Proposition 2 have been obtained under conservative conditions. Furthermore, note that the set of Proposition 2 is asymmetric due to the fact that $e(k) \geq 0$ (delays). There might be the possibilities to obtain narrow sets exploiting mean value analysis. If the delays can be measured, the update law (3) can be easily modified as

$$T_b(k+1) = T_b(k) - \gamma_0 \operatorname{sign}\Big(J(b, T_b(k) + e_b(k)) - J(b, T_b(k-1) + e_b(k-1))\Big)\operatorname{sign}\Big(T_b(k) + e_b(k) - (T_b(k-1) + e_b(k-1))\Big), \tag{9}$$

to obtain the results of Proposition 1 with $\mathcal{A}_b \triangleq [T^\star_b - 2\gamma_0 - e_{\max},\, T^\star_b + 2\gamma_0]$ where $e(k) \leq e_{\max}$ for all $k \geq 0$, which is potentially larger than the set in Proposition 2 but the constraint $|\delta_{T,b}(k)| > \delta_0 \geq |\delta_{e,b}(k)|$ is no longer required.

*Remark 3:* It is noteworthy to highlight the robustness of the AESC algorithm even during transients in which the initial values of $T_b$ might be inappropriate. Unlike previous solutions, e.g., [2], AESC does not probe on the levels of $b$ to estimate the gradient of $J$, which could be very inefficient in presence of significant variations of the arrival rate and if the update of $b$ is relatively too slow. This would led to both very high levels of queued messages when the arrival rate quickly grows, or to high values $J$ when the load quickly drops. On the contrary, assume that with AESC the level of $T_b$ happens to become much lower than the optimal configuration $T^\star_b$ (e.g., due to high measurement errors) and to determine the inappropriate delivery of too small batches. In this case, the sequencer will start to quickly saturate, which implies that the amount of messages unprocessed by the sequencer will quickly grow. These messages will result as immediately available for the subsequent batch, which will consequently be able to buffer very likely a much higher number of messages than in former batches.

In other words, the design choice of associating a level of waiting time $T_b$ with the actual level $b$ of queued messages makes AECS inherently more robust than solutions, such as [2], which perform exploratory steps directly on the batching level $b$, as it will be also confirmed by our experimental study in Section VII.

*Remark 4:* As already hinted, the adaptive probing step of AESC allows to enhance significantly the dynamic performances of our controller. The above convergence proofs clearly still hold in case we use an adaptive gain $\gamma(k)$, defined as

$$\gamma(k) = \begin{cases} \max\left(\gamma_{\min}, \frac{\gamma_0}{b(k)}\right), & \text{if } b(k) > 0, \\ \gamma_0 & \text{elsewhere.} \end{cases} \tag{10}$$

given that $\gamma(k) \in [\gamma_{\min}, \gamma_0]$, as required by Proposition 1 and Proposition 2. The selection $\gamma_0/b(k)$ is quite natural since the increment of the waiting time $T_b(k)$ needs to be smaller as $b$ grows (high levels of $b$ are associated with high

levels of messages arrival rate).

## VII. Evaluation

In this section we evaluate the performance of AESC, providing an assessment of the most important performance metrics for the controller, namely messages self-delivery latency perceived by the application, responsiveness (evaluated in terms of convergence time and relative overshooting amplitude) and stability (evaluated as the ability to cope with strong, abrupt shift of the input signal and in terms of intensity of oscillatory patterns in the output signal).

A fully fledged prototype of AESC has been integrated with the Appia Group Communication System [12], an open source layered communication toolkit fully implemented in Java. In particular, AESC has been designed to be a plug-and-play module for such framework, thus resulting in its employment to be transparent both to the application layer as well as to other modules of the architecture.

The test platform is a cluster composed of 10 machines equipped with two 2.13 GHz Quad-Core Intel(R) Xeon(R) processors and 16 GB of RAM, interconnected via a private Gigabit Ethernet and running the Linux OS 2.6.23-33.

### A. Validating Assumption 1

We start by showing the results of an experimental study aimed at validating Assumption 1 of Section VI, which is a necessary condition underlying the convergence proofs given in Proposition 1 and Proposition 2. In order to do this we measured the self-delivery latency of the system when subject to constant message arrival rates $m$ and statically setting, in each experiment, the batching value $b$ to fixed values in the domain $\{1,2,4,8,16,32,64\}$.

The plots in Fig. 3 show that, once selected an arrival rate $m \in \{1K, 5K, 10K, 15K, 20K\}$ messages per second, the function $J^*(b)$ has a global minimum, thus meaning that, for each arrival rate $m$, a unique optimal batching value $b^*(m)$ exists. Hence, given $b$ and $m$, there exists a unique value $T_b$ that minimizes the self-delivery latency $J(b, T_b)$. The plots also confirm that, for very low arrival rates, batching hampers performance introducing unnecessary latencies and that, at high load, batching is essential to prevent the system from saturating.

### B. Comparison with optimal, statically identified, configurations

Let us now proceed in assessing the performance of AESC, by running the same battery of tests described in the previous section (i.e., generating messages with constant arrival rates), and compare the results obtained using AESC with those produced using a static batching configuration that is optimal for the considered message arrival rate. Fig. 4 reports the results of this study, plotting both the mean self-delivery latency and the mean batching value obtained running the two systems.

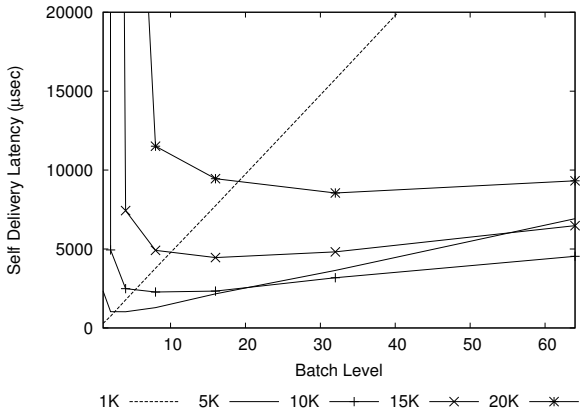


Figure 3. STOB performance while varying mean arrival rate and batching level.

As shown by fig Fig. 4, AESC behaves as well as the the optimal static configuration at very low arrival rates and outperforms it at higher arrival rates, achieving up to a 4x improvement in the case of highest load (20K msgs/sec). This result would clearly be trivial if a variable load had been injected. On the other hand, in this case we used a step signal as input, which makes these results quite surprising.

We argue that the explanation for this phenomenon is that our workload generator introduces some unavoidable fluctuations, whose amplitude grows as the target arrival grows. In other words, the actual input signal is *de facto* not constant, but, rather, it oscillates around the desired target value. This reasoning is clearly backed by observing that the bottom plot of Fig. 5 that reports the dynamics of two step input signals having amplitude of 10K and 20K messages per second. Another confirmation of this argument is obtainable by noticing the improvements achieved by AESC vs a static batching configuration at high load, when the amplitude of the input's signal oscillations is higher.

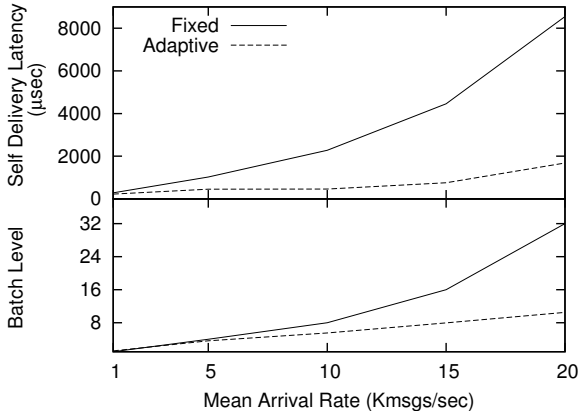


Figure 4. Performance of optimal static configuration vs adaptive tuning.

### C. Dynamic behavior assessment

In this section we describe the dynamic behavior of the controller, by analyzing its response when dealing with varying incoming workloads. The test we designed consists in injecting in the system an arrival rate which is composed of steps of different amplitudes: Fig. 5 depicts two cases in which the message arrival rate is 1000 per second for the first 45 seconds and then it becomes approximately ten or twelve times bigger. The very steep ramp up characterizing the change in the workload allows us to assess the responsiveness and the stability of the system when the batching level is controlled by AESC.

Note that the results plotted in Fig. 5 were obtained using a simple, static (rather than adaptive) exploration step $\gamma = \gamma_0$. We refer to this variant of AESC as to ESC. This choice was done in order to allow isolating the benefits provided by the adaptive exploration step used by AESC.

From the plot, we can see that ESC reacts very quickly to the varying workload. During an initial transitory, the controller exhibits overshoots, moving towards too high batching levels. This is caused by the fact that the update of the waiting time $T_b$ is performed each time via the same $\gamma_0$, which is very big compared to the inter-arrival time of the messages when the arrival rate is high. Then, the controller converges to the optimal value, slightly oscillating around it till the end of the test. The short overshooting period is due to the fact that ESC is forced to operate in zones still unexplored, whose values for $T_b$ have been initialized with suboptimal values relevant to the only states excited so far. This yields to an initial accumulation of messages in the queue, which in turn induces an abrupt increase of the self-delivery latency. To compensate it, the controller operates a series of consecutive exploration steps aimed at increasing the level of batching and quickly serve all the messages in the queue; in the meanwhile, the controller moves to the optimal batching value.

Let us now observe the results reported in Fig. 6, which contrast the response to two subsequent step signals having amplitude 20K msgs/sec when using AESC and ESC. The plots allow to draw two interesting conclusions.

First, the adoption of the adaptive probing step, in AESC, allows to significantly reduce the amplitude of the overshooting phenomenon that affects ESC. This is a direct consequence of the possibility of setting a finer grained exploration step for high batching values, which allows to improve significantly the speed of convergence towards the optimal batching value at high load levels.

Second, the stateful nature of AESC, which maintains a memory of the optimal batch waiting time for the various feasible batch values $b \in \mathcal{B}$, allows to avoid the occurrence of overshooting phenomena when the second step is generated in input (at time 125 sec). In this case, in fact, AESC can take advantage of having already learnt the optimal batching value for this load level during the first step input signal (which took place, approximately, since second 50 until second 90).

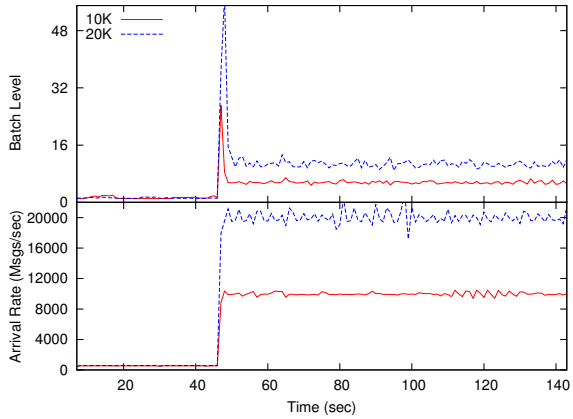


Figure 5. Dynamic response of the adaptive scheme to step input signals.

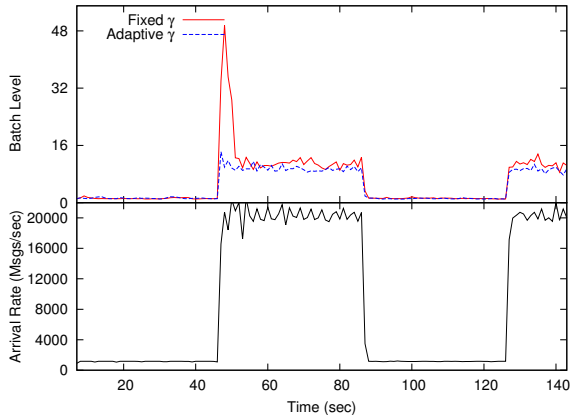


Figure 6. Dynamic response of the adaptive scheme to step input signals.

### D. Determining $\gamma_0$ and $\gamma_{min}$

Let us now discuss a possible methodology for determining adequate values for the parameters $\gamma_0$ and $\gamma_{min}$. The setting of these two parameters is clearly an important choice, as they drive the research for the optimal value $T_b^\star$ and, as stated in Proposition 2, they contribute to define the amplitude of the oscillation around it. It is, thus, important to pick values that ensure that exploratory steps are large enough at low load to guarantee the actual exploration of higher batching levels, but also small enough to avoid ample oscillations around the optimal $T_b^\star$ for high message arrival rates.

In order to automatically detect the right value for $\gamma_0$, it is sufficient to run a simple benchmark that injects messages at increasing rates, with batching values set to one and two, till it determines the lowest arrival rate $m$ at which the system benefits from increasing the level of batching. The inverse of this value represents the minimum amount of time the

controller has to wait to create, on average, a batch of size 2 in presence of the lowest load level that would benefit from such a batching level configuration. We found this arrival rate to be about 4000 messages per second for our experimental test-bed, thus yielding to $\gamma_0 = 250\mu sec$.

The dual reasoning applies to $\gamma_{min}$: via an initial benchmarking phase the maximum throughput achievable by the system (using the maximum batching value) is determined, and its inverse is chosen as minimum step to guarantee explorations also at very high loads. In our test case, we found $\gamma_{min}$ to be 50 $\mu sec$.

### *E. Comparison with approximate gradient descent algorithm*

In this subsection we compare the performance of AESC with that of a simple approximate gradient descent algorithm, denoted as AGD in the following, which closely resembles the one presented in [2]. This algorithm attempts, at each iteration, to alter the batching value (increasing or decreasing it by one) on the basis of feedback gathered monitoring the same performance index used by AESC, namely the self-delivery latency measured at the sequencer side. This simpler, exploration-based self-tuning technique differs from AESC in two fundamental aspects:

- it attempts to optimize directly the level of message batching $b$, unlike AESC, which conversely optimizes the batch waiting time;
- it is "stateless", in the sense that it does not maintain any memory of the optimal batching values identified in the past, while being subject to different load levels.

The plots in Fig. 7 and Fig. 8 report the results of a sensibility analysis in which we vary the filtering level, namely the time after which the performance index is deemed to have reached steady state, and is used to convey a feedback to the controller. In this case, in fact, we found out, experimentally, that, given the design choice of optimizing directly the batching level, this solution is extremely sensitive to the setting of this parameter.

Specifically, the plots in Fig. 7 highlight that filtering can have a beneficial impact on the stability of the controller, avoiding to incur in strong fluctuations that instead affect the algorithm in case no filtering is used. On the down side, filtering clearly introduces latencies that hinder the responsiveness of the controller.

The results presented in Fig. 8, on the other hand, show that, when faced with very abrupt variations of the input signal, the latencies introduced by the filtering can severely affect the convergence of this algorithm toward the optimal solution. Indeed, if we use input step signals having amplitude equal to 20K messages per second, the only configuration in which AGD succeeds in sustaining the load is the one in which filtering is disabled. When filtering is on, the latencies it introduces do not allow the controller to react sufficiently fast to the quick load surge,

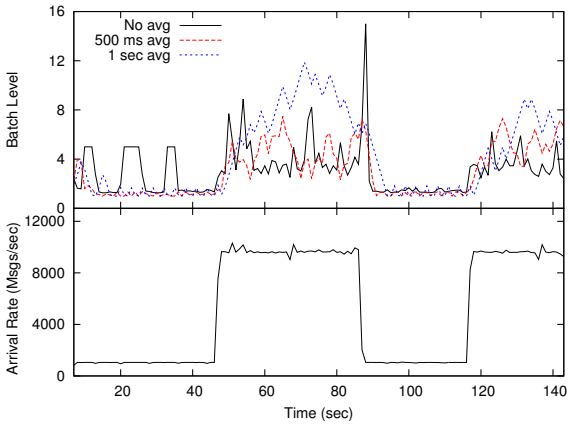


Figure 7. Dynamic response of the AGD algorithm with 10K msgs/sec input step signals, while varying the filtering level.

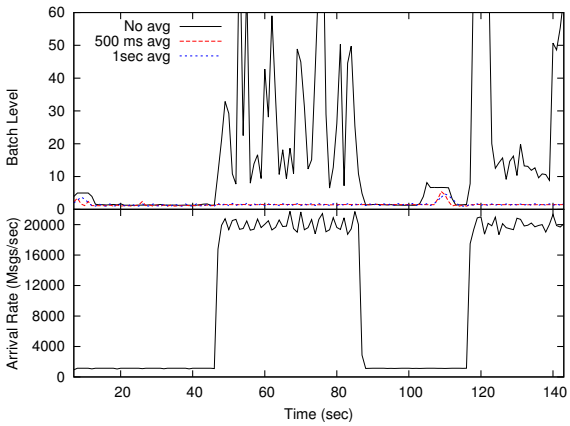


Figure 8. Dynamic response of the AGD algorithm with 20K msgs/sec input step signals, while varying the filtering level.

rendering it unable to ever converge, and leading the group communication system to eventually collapse.

Finally, we conclude by contrasting directly the self-delivery latency perceived when using AGD (without filtering) and AESC, in presence of input signals of amplitude 20K messages per second. These plots clearly highlight the superior performance achieved by AESC, which ensures self-delivery latencies that are, on average, one order of magnitude smaller that for AGD.

## VIII. Conclusions

We have proposed an Adaptive Extremum Seeking Controller (AESC) to optimize the performance of the Sequencer-based Total Order Broadcast (STOB). The controller performs (approximated) gradient-like searches to continuously select the best waiting times $T_b$ associated to each batching level $b$. Stability results on the convergence domain of the algorithm have been given under different conditions and its effectiveness has been shown via extensive experimental evaluation through the Appia Group System.

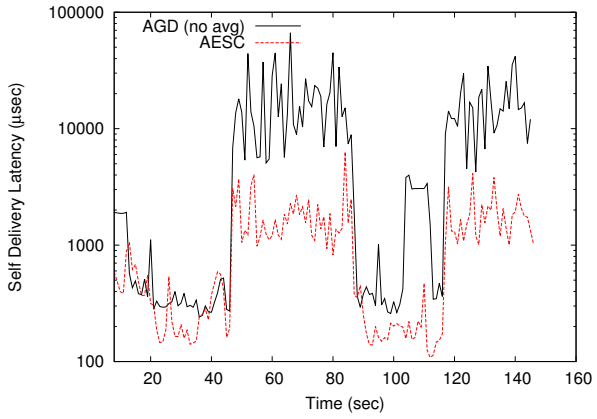


Figure 9. Comparing the dynamic response time (self-delivery latency) of AGD and AESC, with 20K msgs/sec input step signals

## ACKNOWLEDGMENTS

This work has been partially supported by the project "Cloud-TM" (co-financed by the European Commission through the contract no. 257784), and by FCT Fundação para a Ciência e a Tecnologia, under project ARISTOS (PTDC/EIA-EIA/102496/2008) and PEst-OE/EEI/LA0021/2011.

## REFERENCES

[1] R. Friedman and E. Hadad, "Adaptive batching for replicated servers," in *Proc. of the IEEE International Symposium on Reliable Distributed Systems (SRDS)*, 2006.

[2] A. Bartoli, C. Calabrese, M. Prica, E. A. D. Muro, and A. Montresor, "Adaptive message packing for group communication systems," in *Proc. of the Workshop on Reliable and Secure Middleware (WRSM)*, 2003.

[3] B. Carmeli, G. Gershinsky, A. Harpaz, N. Naaman, H. Nelken, J. Satran, and P. Vortman, "High throughput reliable message dissemination," in *Proc. of the Symposium On Applied Computing (SAC)*, 2004.

[4] T. Friedman and R. V. Renesse, "Packing messages as a tool for boosting the performance of total ordering protocols," in *Proc. of the International Symposium on High Performance Distributed Computing (HPDC)*, 1997.

[5] D. Powell (ed.), *Special Issue on Group Communication*. ACM, 1996, vol. 39, no. 4.

[6] K. Ostrowski, K. Birman, and D. Dolev, "Live distributed objects: Enabling the active web," IEEE Internet Computing, 2007.

[7] P. Romano, D. Rughetti, F. Quaglia, and B. Ciciani, "Apart: Low cost active replication for multi-tier data acquisition systems," in *Proc. of the IEEE International Symposium on Network Computing and Applications (NCA)*, 2008.

[8] F. Pedone, R. Guerraoui, and A. Schiper, "The database state machine approach," *Distributed and Parallel Databases*, vol. 14, no. 1, 2003.

[9] N. Carvalho, P. Romano, and L. Rodrigues, "Asynchronous lease-based replication of software transactional memory," in *Proc. of the ACM/IFIP/USENIX International Middleware Conference (Middleware)*, 2010.

[10] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, 1998.

[11] ——, "Lower bounds for asynchronous consensus," *Distributed Computing*, vol. 19, no. 2, 2006.

[12] A. Pinto, "Appia: A flexible protocol kernel supporting multiple coordinated channels," in *Proc. of the International Conference on Distributed Computing Systems (ICDCS)*, 2001.

[13] J. Nagle, "Congestion control in ip/tcp internetworks," *ACM SIGCOMM Computer Communication Reviews*, vol. 14, 1984.

[14] Paolo Romano, Matteo Leonetti, "Self-tuning Batching in Total Order Broadcast Protocols via Analytical Modelling and Reinforcement Learning," in *Proc. of the IEEE International Conference on Computing, Networking and Communications, Network Algorithm & Performance Evaluation Symposium (ICNC)*, 2012.

[15] F. Cristian, R. D. Beijer, and S. Mishra, "A performance comparison of asynchronous atomic broadcast protocols," *Distributed Systems Engineering*, vol. 1, 1994.

[16] R. Ekwall and A. Schiper, "Modeling and validating the performance of atomic broadcast algorithms in high-latency networks," in *Proc. of the International European Conference on Parallel and Distributed Computing (Euro-Par)*, 2007.

[17] A. Coccoli, P. Urban, A. Bondavalli, and A. Schiper, "Performance analysis of a consensus algorithm combining stochastic activity networks and measurements," in *Proc. of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2002.

[18] N. Santos and A. Schiper, "Tuning paxos for high-throughput with batching and pipelining," in *Proc. of the International Conference on Distributed Computing and Networking (ICDCN)*, 2012.

[19] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to tcp throughput prediction," in *Proc. of the ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[20] L. Garces-Erice, "Admission control for distributed complex responsive systems," in *Proc. of the International Symposium on Parallel and Distributed Computing (ISPDC)*, 2009.

[21] M. Couceiro, P. Romano, and L. Rodrigues, "A machine learning approach to performance prediction of total order broadcast protocols," in *Proc. the IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2010.

[22] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, no. 9-12, 2007.

[23] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches," *Cluster Computing*, vol. 11, no. 3, 2008.

[24] A. Andrzejak and L. Silva, "Using machine learning for non-intrusive modeling and prediction of software aging," in *Proc. of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2008.

[25] R. Guerraoui and L. Rodrigues, *Introduction to Reliable Distributed Programming*, 2006.

[26] C. Drapper and Y. Li, "Principles of optimalizing control systems and an application to the internal combustion engine," *ASME*, vol. 160, 1951.

[27] I. Morosanov, "Method of extremum control," *Automation and Remote Control*, vol. 18, 1957.

[28] M.Krstić and H.H.Wang, "Stability of extremum seeking feedback for general nonlinear dynamic systems," *Automatica*, vol. 36, 2000.

[29] K. Ariyur and M. Krstić, *Real-Time Optimization by Extremum-Seeking Control*, 2003.

[30] M. Guay, D. Dochain, and M. Perrier, "Adaptive estremum seeking control of continuous stirred tank bioreactors with unknown growth kinetics," *Automatica*, vol. 40, no. 5, 2004.

[31] K. Peterson and A. Stefanopoulou, "Extremum seeking control for soft landing of an electromechanical valve actuator," *Automatica*, vol. 40, 2004.

[32] M. Sassano, D. Carnevale, and A. Astolfi, "Extremum seeking-like observer for nonlinear systems," in *Proc. of the IFAC World Congress*, vol. 18, 2011.

[33] D. Carnevale, A. Astolfi, L. Zaccarian, L. Boncagni, C. Centioli, S. Podda, and V. Vitale, "Maximizing radiofrequency heating on ftu via extremum seeking: parameter selection and tuning," *Book chapter, "From physics to control through an emergent view"*, vol. 15, 2010.

[34] Y. Tan, D. Nesic, and I. M. Y. Mareels, "On non-local stability properties of extremum seeking control," *Automatica*, vol. 42, no. 6, 2006.

[35] D. Nešić, Y. Tan, W. H. Moase, and C. Manzie, "A unifying approach to extremum seeking: Adaptive schemes based on estimation of derivatives," in *Proc. of the IEEE Conference on Decision and Control*, 2010.

[36] L. Fu and ı. Özgüner, "Extremum seeking with sliding mode gradient estimation and asymptotic regulation for a class of nonlinear systems," *Automatica*, vol. 47, no. 12, 2011.

[37] A. Teel and D. Popovic, "Solving smooth and nonsmooth multivariable extremum seeking problems by the methods of nonlinear programming," in *Proc. of the American Control Conference*, 2001.

[38] J. Nocedal and S. Wright, *Numerical Optimization*, 1999.

[39] I. K. Argyros, *Convergence and application of Newton-type Iterations*, 2008.

[40] C. D. Fiore, S. Fanelli, and P. Zellini", "Low complexity secant quasi-newton minimization algorithms for nonconvex functions," *Journal of Computational and Applied Mathematics*, vol. 210, no. 12, 2007.

[41] H. K. Khalil, *Nonlinear systems*, 2002.