A time synchronization protocol for large-scale distributed embedded systems with low-precision clocks and neighbor-to-neighbor communications

Andre Naz, Benoit Piranda, Julien Bourgeois, Seth Copen Goldstein

A Time Synchronization Protocol for Large-Scale Distributed Embedded Systems with Low-Precision Clocks and Neighbor-to-Neighbor Communications

André Naz[a,*], Benoît Piranda[a], Julien Bourgeois[a], Seth Copen Goldstein[b]

[a] *Univ. Bourgogne Franche-Comté (UBFC), FEMTO-ST Institute, CNRS, 25200 Montbéliard, France*
[b] *Carnegie Mellon University*
Pittsburgh, PA 15213, USA

Abstract

In this paper, we propose the Modular Robot Time Protocol (MRTP), a network-wide time synchronization protocol for modular robots (a class of distributed embedded systems) with neighbor-to-neighbor communications and potentially low-precision clocks. Our protocol achieves its performance by combining several mechanisms: central time master election, fast and recursive propagation of synchronization waves along the edges of a breadth-first spanning-tree, low-level timestamping and per-hop compensation for communication delays using the most-appropriate method, and clock skew compensation using linear regression. We evaluate our protocol on the Blinky Blocks system both on hardware and through simulations. Experimental results show that MRTP can potentially manage real systems composed of up to 27,775 Blinky Blocks. We observe that the synchronization precision depends on the hop distance to the time master, the synchronization periods and the number of synchronization points used for the linear regressions. Furthermore, we show that our protocol is able to keep a Blinky Blocks system synchronized to a few milliseconds, using few network resources at runtime, even-though the Blinky Blocks hardware clocks exhibit very poor accuracy and resolution. We compare MRTP to existing synchronization protocols ported to fit our system model. Simulation results show that MRTP can achieve better synchronization precision than the most precise compared protocols while sending more than half less messages in compact systems.

Keywords: Time synchronization, Distributed embedded systems, Modular robotics

1. Introduction

Technological advances, especially in the miniaturization of robotic devices, foreshadow the emergence of large-scale ensembles of small-size and resource-constrained robots that will self-organize and distributively cooperate to achieve complex tasks (e.g., modular robotic systems (Yim et al., 2009), swarm robotic systems (Şahin, 2004), distributed micro-electro-mechanical systems (Bourgeois and Goldstein, 2012), robotic materials (McEvoy and Correll, 2015), programmable matter (Goldstein and Mowry, 2004; Bourgeois et al., 2016), etc.). These ensembles are formed from independent, intelligent and communicating robots which act as a whole. In addition, these robots are able to re-arrange their connectivity in order to adapt the global structure of their ensemble to specific applications, providing robustness and versatility.

We believe that lots of innovative and complex applications based on large-scale ensembles of robots are to appear and will be fully integrated in our daily-life environment. In (Goldstein and Mowry, 2004; Bourgeois et al., 2016), it is, for example, envisioned to use thousands to millions of micro modules to build programmable matter, i.e., matter that can change its physical properties in response to external and programmed events.

*Corresponding author
Email addresses: andre.naz@femto-st.fr (André Naz), benoit.piranda@femto-st.fr (Benoît Piranda), julien.bourgeois@femto-st.fr (Julien Bourgeois), seth@cs.cmu.edu (Seth Copen Goldstein)

Programmable matter promises synthetic reality and has a wide range of applications (e.g., sending/downloading copies of physical objects, morpheable objects re-shapable at our will, injectable surgical instruments, 3-D interactive life-size TV, etc.) (Goldstein and Mowry, 2004).

Coordination among a group of distributed robots often rely on time synchronization. For instance, (Habibi et al., 2012) proposes to attach several modular robots together, each one equipped with a conveyance surface, to build a large distributed and modular system to convey small and fragile objects. Modules cooperate to convey the object using distributed real-time control. Modules have to remain synchronized in order to satisfy timing constraints, otherwise the object may get out of the trajectory, hit obstacles or fall off the surface. A distributed self-reconfiguration algorithm in which cubic modules arranged in a 2D plane cooperatively move to change the global organization of the modular robot is proposed in (Piranda and Bourgeois, 2016). In this algorithm, lines of modules are translated. A line translation requires modules in that line to be synchronized in order to move quasi-simultaneously in the same direction. In (Hosseinmardi et al., 2012), the authors propose to realize amorphous intelligent façades composed of many actuating sensor/actuator cells, each one equipped with RGB leds. One can imagine to use this façade as a large-scale distributed display in which every cell is a pixel and the cells collaboratively render images and sound in a synchronous fashion. Coordination of these robots will require synchronized clocks.

In this paper, we focus our attention on modular robots composed of identical modules that communicate together using neighbor-to-neighbor communications. The term "neighbors" designates neighbors in the physical network and refers to directly connected modules (one-hop). Modules can share a common timing signal through dedicated pins, but this requires a specific hardware design. We consider a system without a global clock signal. Every module has its own notion of time provided by its own hardware clock. Since common hardware clocks are imperfect, local clocks tend to run at slightly different and variable frequencies, drifting apart from each other over time. Consequently, a distributed time synchronization is necessary to keep modules local clock synchronized. The offset of two clocks denotes the time difference between them whereas the skew of two clocks denotes their frequency difference.

Several approaches to time synchronization exist (continuous vs on-demand, network-wide vs clustering, timescale transformation vs clock synchronization, etc.) (Römer et al., 2005). The approach to use depends on the target application. In the continuous model, nodes strive to kept synchronized at all time. This model is opposed to the on-demand synchronization model where nodes can either a posteriori agree on the time at which an event has occurred or anticipate synchronizations in order to trigger some coordinated actions at a given time. Here, our goal is to achieve network-wide and continuous time synchronization. This is indeed the most general approach, in our opinion.

Synchronization protocols based on this approach aim to keep a small offset between local clocks and a global reference time. In most of the existing protocols, devices exchange timestamped messages in order to estimate the current global time. Since time keeps going during communications, modules have to correctly compensate for network delays in order to evaluate the current global time upon reception of synchronization messages. Although it is non-trivial to accurately estimate communication delays, especially in presence of unpredictable delays (due for example, to queueing or retransmissions), it is crucial in order to achieve high precision performance.

Our contribution is to propose the Modular Robot Time Protocol (MRTP), a network-wide time synchronization protocol for modular robots. This work is an extension of our previous one presented in (Naz et al., 2015a, 2016b). In (Naz et al., 2015a, 2016b), we evaluate the precision of MRTP in small-scale hardware systems and we announce by extrapolation that MRTP can efficiently manage systems with up to $27,775$ modules. The actual contribution of this article is twofold. First, we build an accurate simulation model of our target platform and evaluate the performance of MRTP on larger-scale systems through simulations. We consider various performance metrics, namely, the synchronization precision, the time of convergence and the communication efficiency. Second, we compare MRTP to different existing synchronization protocols. We show in Section 6.2 that MRTP can effectively synchronize systems with up to $27,775$ modules to a few milliseconds. Moreover, simulations results show that MRTP can achieve better synchronization precision than the most precise compared protocols while sending more than half less

messages in compact systems.

MRTP is intended to synchronize large-scale and fairly stable systems where changes in the network topology, due for instance to module mobility, or potential module or link failures, are infrequent. Our protocol assumes every module has a local clock, which can be low precision and low resolution, typically in the order of the millisecond. Modules can use low communication bitrates (e.g, 38.4 $kbit/s$). Such a low resolution, low precision and high communication latency make accurate synchronization challenging. First, the local time cannot be accurately read. Second, it is hard to accurately compensate for network delays if they are not negligible and, at the same time, only roughly measurable. Third, clock skew and clock instability may not be negligible during high-latency (multi-hop) communications.

To the best of our knowledge, MRTP is the first protocol for modular robots that provides an accurate low-skew global timescale without dedicated hardware. Our protocol combines new ideas with existing methods proposed in the domains of computer networks and wireless sensor networks. In our protocol, a dynamically elected central module periodically broadcasts the current global time along the edges of a breadth-first spanning-tree. Placing the time master close to the network center of the system reduces the time of the synchronization phases and increases the overall precision as cumulative estimations are made every hop. It is particularly efficient in our context, because, as explained in Section 3.1, large-scale ensembles of modular robotic ensembles where modules use neighbor-to-neighbor communications tend to exhibit large hop distances. At each hop, the propagated estimation of the current global time is updated to take into account communication delays and time of residence in intermediate modules. Any approach to compensate for these delays can be used in MRTP. Most of the existing approaches use low-level timestamping to suppress the main sources of uncertainty in delay estimations. The best-suited technique to use in MRTP depends on the target platform (i.e., the clock precision, its resolution and the communication mechanism) and should be carefully selected as it has a direct impact on the performance of our protocol. Section 6.1.3 provides a method to experimentally evaluate the precision of a given approach over multiple hops. In the system we use for our experiments, the most precise method is based on data-link layer timestamping and predictions of the transfer time (as defined in Section 3.3). With this approach, a module gets synchronized by a single timestamped message from its parent in the synchronization tree incurring little message overhead. Furthermore, modules use linear regression to compensate for clock skew.

We implemented our protocol and evaluated it on the Blinky Blocks system (Kirby et al., 2011), both on hardware[1] and in the VisibleSim simulator[2] (Dhoutaut et al., 2013). We provide two videos of MRTP running on the Blinky Blocks hardware platform to illustrate the necessity, the effectiveness and the robustness of MRTP[3,4]. In this article, we show that MRTP is able to manage systems composed of up to 27,775 Blinky Blocks. Furthermore, experimental results show that MRTP is able to successfully maintain a Blinky Blocks system synchronized to a few milliseconds, using few network resources at runtime, although the Blinky Blocks use low-bitrate communications (38.4 $kbit/s$) and are equipped with a very low accuracy (10,000 parts per million (ppm)) and poor resolution (1 millisecond) clocks.

The rest of this paper is organized as follows. Section 2 offers an overview of the existing time synchronization protocols. Section 3 details the system model and assumptions. Section 4 describes MRTP. Section 5 describes the technical characteristics of the Blinky Blocks, the target platform. Section 6 presents experimental results. Section 7 concludes our work and discusses its general applicability. Section 8 suggests future research directions.

2. Related Work

Time synchronization has been extensively studied in various sub-domains. Many algorithms

[1] The source code of MRTP is included in the Blinky Blocks firmware, available online at https://github.com/claytronics/oldbb

[2] The source code of VisibleSim and the applications written for the evaluation of our protocol is available online at https://github.com/nazandre/thesis

[3] A video illustrating the necessity of time synchronization in Blinky Blocks systems and the robustness of MRTP over time is available online at: https://youtu.be/X6QzivsmJBo

[4] A video illustrating the effectiveness of MRTP and its robustness to network dynamics is available online at: https://youtu.be/xpbnbeJVz08. In this video, 72 Blinky Blocks are assembled to build a distributed and extensible bitmap scroller.

and protocols have been proposed for computer networks such as Cristian's algorithm (Cristian, 1989), the Berkeley algorithm (Gusella and Zatt, 1989), the Network Time Protocol (NTP) (Mills, 1991) and the IEEE 1588 Precise Time Protocol (PTP) (IEEE, 2008). Time synchronization is also an important topic of interest in Wireless Sensor Networks (WSN) where many protocols have been proposed, e.g., Reference Broadcast Synchronization (RBS) (Elson et al., 2002), the Timing-sync Protocol for Sensor Networks (TPSN) (Ganeriwal et al., 2003), the Flooding Time Synchronization Protocol (FTSP) (Maróti et al., 2004), the Time-Diffusion Synchronization Protocol (TDP) (Su and Akyildiz, 2005), the Rapid Time Synchronization (RATS) (Kusy, 2007), the PulseSync (Lenzen et al., 2009, 2015), the Asynchronous Diffusion algorithm (AD) (Li and Rus, 2006), the Gradient Time Synchronization Protocol (GTSP) (Sommer and Wattenhofer, 2009), the Average TimeSynch (ATS) protocol (Schenato and Fiorentin, 2011) and the Maximum Time Synchronization (MTS) (He et al., 2014a). Like modular robots, WSN generally form spontaneous peer-to-peer networks of resource constrained devices.

To the best of our knowledge, time synchronization has not attracted any attention in the modular robotic community. Methods to provide a global metronome-like signal in modular robots have been proposed in (Kokaji et al., 1996; Baca et al., 2010). However, these mechanisms synchronize clock phase or/and frequency but not actual clock time. Moreover, (Kokaji et al., 1996) is purely theoretical, the authors consider ideal clocks running at the same exact frequency and do not provide any performance evaluation.

2.1. Architecture : from master/slave to fully distributed protocols

Existing time synchronization protocols differ by the network architecture they adopt. NTP, PTP, TPSN, FTSP, RBS and TDP adopt a master/slave approach. In a master/slave approach, one or more masters are in charge of synchronizing slave nodes. In NTP, PTP, TPSN, FTSP and TDP, slave node clocks are adjusted to a reference time held by the time master(s). The reference time can be the Coordinated Universal Time or the master local clock. In the Berkeley algorithm, slave node clocks are adjusted to an aggregated value of some or all the system clock values. These approaches aim at performing global synchronization, i.e., keeping all nodes synchronized together. These protocols provide a satisfactory synchronization precision between arbitrary nodes but may poorly synchronize neighboring nodes. This is due to the fact that two neighboring nodes can be synchronized by messages that have traveled on almost independent paths causing the error accumulated at every hop to be propagated differently.

In contrast, AD, ATS, GTSP and MTS are fully distributed. In these protocols, nodes exchange timing information with all their one-hop neighbors on a regular basis. In AD, every node frequently adjusts its clock to the average value of its neighbors' clock. ATS and GTSP use a similar consensus-based averaging technique. These average-based approaches primarily aim at achieving local synchronization, i.e., keeping neighboring nodes synchronized together, allowing nodes to have a larger pairwise synchronization error with nodes that are far away. MTS and its variants proposed in (He et al., 2014a,b) use extremum-value based consensus to achieve faster convergence. In general, fully distributed methods are naturally fault tolerant and robust to node mobility. However, they can lead to a long convergence time and to a high message complexity, especially in point-to-point networks without broadcast support. Indeed, in systems without local nor global shared broadcast medium, a node has to send individual messages to all neighbors in order to broadcast messages.

2.2. Infrastructure of master/slave protocols

Master/slave time synchronization protocols differ by the infrastructure they use. Protocols can use tree-like structures, cluster-based structures or be infras-tructure-less.

Tree-like structures. NTP, PTP and TPSN use tree-like hierarchical structures rooted at the time master(s) to spread timing information. Logical neighbors in the tree(s) can be neighbors in the physical network as in TPSN, or potentially distant as in NTP. The latter case may require multi-hop communications that rely on the existence of an underlying routing service. In our case, we assume no routing service. In TPSN, nodes are recursively synchronized hop-by-hop along the edges of the synchronization tree starting from the time master. Hence, during each synchronization phase, the current global time gets quickly disseminated through the entire network. In addition to provide a relatively quick synchronization convergence,

this reduces the impact of clock inaccuracies (due to noise, skew variations, time-increasing errors in the local estimation of the global time) on the synchronization process.

Clustering based on broadcast domains. In RBS, nodes maintain relative timescales of their neighborhood using reference pulses broadcasted by some master nodes. In multi-hop networks, nodes can be grouped into overlapping clusters based on broadcast domains and border nodes act as gateways to translate clock values.

Infrastructure-less approaches. In contrast, FTSP, RATS and PulseSync are infrastructure-less. They provide robustness to network topology changes and to link failures using either periodic local broadcasts or periodic network-wide floodings. In FTSP, the time master and the synchronized nodes periodically broadcast their estimation of the current global time to all their neighbors, in an asynchronous fashion. Synchronization waves propagate with a limited speed through the network. Indeed, after having received a new synchronization message, a node has to wait until the expiration of its broadcast period to transmit the information to its neighbors. As a consequence, the time-increasing estimation error of the global time is amplified at every hop and FTSP exhibits a synchronization error that grows exponentially with the size of the network (Lenzen et al., 2009). Hence, optimal synchronization requires fast network flooding (Lenzen et al., 2009). RATS and PulseSync employs rapid network-wide floodings using recursive broadcasts to quickly disseminate the global time through the network. The time master periodically launches synchronization waves using broadcasts. Slave modules re-broadcast new synchronization messages shortly after reception. In reliable and fairly static point-to-point networks without broadcast support, recursive synchronization using a tree-like structure is more communication-efficient than network-wide flooding.

2.3. Communication delay compensation methods

Time synchronization protocols also differ by the methods they use to compensate for communication delays. The method to use depends on the target platform and directly impacts the precision of the synchronization protocol. Existing methods can be divided in three categories: approaches based on the round-trip time, methods based on byte-level timestamping and approaches based on reference broadcasts.

Round-trip time based methods. Cristian's algorithm, the Berkeley algorithm, NTP, PTP, TPSN and TDP measure half the round-trip time to estimate one-way communication delays. Cristian's algorithm and NTP perform end-to-end synchronization on possibly multi-hop paths. They use statistical analysis to mitigate variation in delays do to retransmission(s), queueing, route selection, etc. These methods are expensive in communications and in computations. PTP and TPSN propose to perform per-hop synchronization with low-level timestamping to prevent unpredictable delays induced at the different layers of the network stack from affecting delay measurements. PTP can use timestamps recorded at the physical layer to achieve high accuracy if dedicated hardware is available. In TPSN, a node exchanges a single bidirectional message timestamped at the boundary of the data-link layer to synchronize itself to another node and compensates for communication delays using half of the round-trip time. We call this method RTT (for Round-Trip Time). Round-trip time methods assume symmetrical nominal delays and neglect the effect of clock skew during the round-trip.

Byte-level timestamp based methods. FTSP, PulseSync and the practical implementations of ATS (Schenato and Fiorentin, 2011) and MTS (He et al., 2014b) use byte-level time-stamping, which requires an intimate access to the data-link layer.

In the last two, nodes exchange a single unidirectional message timestamped just before the transmission of the first byte (i.e., the Frame Delimiter byte) and upon reception of this byte. The time elapsed between the transmission and the reception of the frame delimiter byte is neglected and ATS / MTS consider that the two timestamps refers to the same real time. We call this method FD (for Frame Delimiter). This method neglects the interrupt handling time, the frame delimiter byte transmission / reception, the propagation time and the time required to detect the frame delimiter byte. Although FD works well in low-latency networks, the neglected time can be important in higher-latency systems. For instance, our target system uses 38.4 $kbit/s$ connections while WSN that use IEEE 802.11b communications have a maximal bitrate of 11 $Mbit/s$. At 38.4 $kbit/s$, a byte is transmitted in roughly 208μs while at 11 $Mbit/s$ a byte

is sent in less than 1 μs.

FTSP goes one step further in order to eliminate most of the sources of delays in message transmission (except the propagation time). FTSP synchronizes neighbors using a single message broadcast with statistical operations on timestamps captured at the byte boundary during interrupts at the data-link layer. The latest version of PulseSync (Lenzen et al., 2015) is based on an enhanced version of the FD method. The authors use the slotted programming approach (Flury and Wattenhofer, 2010) to minimize interrupt latency and use a static value measured experimentally during a calibration phase to compensate for the time between the insertion of the timestamp, just before transmitting the frame delimiter byte, and its detection upon reception. However, the method proposed in FTSP and PulseSync can not be applied directly to our target system. Indeed, we assume low-resolution clocks, typically in the order of the millisecond, that can not efficiently capture phenomena at the byte transmission level which occurs at the microsecond scale.

Reference broadcast. In RBS, some reference nodes periodically broadcast reference messages. Neglecting propagation delays, receiving nodes use the data-link reception times as reference points to compare their clock values all together. This requires a shared broadcast medium and it is not usable in point-to-point networks.

Discussion. We argue that the method to compensate for communication delays has to be selected in function of the target system. If we assume predictable transfer time between neighboring modules, we propose to perform per-hop synchronization using a single unidirectional message timestamped at the data-link layer and predictive communication delay compensation (see Section 3.4). We call this method PRED (for Predictive). We show in Section 6.1.3 that, in our target system, PRED is in average more precise than the two other methods which can be applied to our target system, namely FD and RTT. Note that this is mainly due to the fact that the average transfer delay of a frame is almost a round number (at the millisecond scale) in this system.

2.4. Clock model: from clock offset adjustment only to clock skew compensation

Furthermore, time synchronization protocols differ in the clock model they use. In some protocols, e.g., AD and TPSN, nodes perform clock offset adjustment only and do not take into account clock skew. Compensation for clock skew enables modules to be synchronized less frequently without degrading the synchronization precision.

NTP uses phase-locked loops and/or frequency-locked loops. In (Kim et al., 2012), the authors use a Kalman filter to track clock offset and skew with low-precision oscillators and time-varying skew. Indeed, in the presence of ambient environment variations (e.g., temperature variations), the clock skew may vary over-time.

ATS, BP (Etzlinger et al., 2014), GTSP, MF (Etzlinger et al., 2014), MTS, FTSP, RBS, PulseSync, RATS and (Noh et al., 2007; Leng and Wu, 2010) propose to model clock using a linear model computed on recent observations, assuming that oscillators have high short-term stability. Indeed, if we assume that environment changes do not happen or happen gradually, clock skew will change smoothly. RBS, FTSP, PulseSync and RATS use least-square linear regression on recent window of observations. ATS and GTSP use averaging technique to estimate clock skew based on the previous synchronization points. In (Noh et al., 2007; Leng and Wu, 2010), the authors propose to enhance TPSN by using a linear model and maximum likelihood estimators. BP and MF derive maximum a posteriori estimators of the clock parameters using respectively, belief propagation and mean field on factor graphs. Different methods for clock skew compensation including linear regression, exponential averaging and phase-locked loops have been evaluated in (Amundson et al., 2008). Although results are nearly identical, experiments suggest that linear regression leads to slightly more precision.

Note that in addition to compensate for clock skew, these aggregating techniques also tend to reduce the impact of the measurement errors.

2.5. Time master election

Master/slave time synchronization protocols also differ by the mechanisms they employ to select the time master. In NTP and RATS, time masters are pre-configured. In our case, it is more flexible if the system itself elects its time master. In PTP and TDP, elections are based on the quality of the clocks. In addition, TDP periodically re-elects time masters to balance the load. FTSP and PulseSync implicitly elect the minimum-identifier node as the time master during the synchronization phases.

In our case, we consider systems where all modules are identical and equipped with the same hardware clocks. Although these clocks slightly differ in their accuracy and stability (Naz et al., 2016b), we consider that with a careful selection of the hardware, the impact of cumulative errors made in network delay compensations will be predominant in large-diameter systems. Indeed, a random error is experienced at each hop. If we us assume that these per-hop errors are independent and identically distributed with a mean of λ and a standard-deviation of σ, then the Central Limit theorem states that the error accumulated over k hops follows a normal distribution with a mean of λk and a standard-deviation of $\delta\sqrt{k}$. Experimental results presented in Section 6.1.3 confirm this trend. Hence, we propose to elect a central module as the time master.

2.6. Summary

Table 1 summarizes the related work. Existing protocols contain interesting ideas but fail to efficiently adapt to modular robot systems where modules use low-bitrate neighbor-to-neighbor communications, hardware clocks have low precision and the network diameter can be large. In the absence of a (locally) shared communication medium, infrastructure-less approaches are too expensive in terms of communication compared to tree-based approaches. The method to compensate for network delays has to be carefully selected in function of the target platform. Furthermore, criteria considered for time master election are not adapted to modular robots running under our assumptions. Node centrality can be considered for the election in order to increase the overall synchronization precision.

3. System Model and Assumptions

3.1. General system model

In this article, we consider modular reconfigurable robots that form asynchronous non-anonymous point-to-point connected networks in which modules use neighbor-to-neighbor communications. We assume every module has a unique identifier and maintains a consistent list of its neighbors using an external algorithm. Module identifiers are used in the suggested external algorithms to elect the time master (see Section 4.2.1). The network has neither local nor global shared broadcast medium. In particular, to broadcast a message, a module has to send an individual copy of that message to all neighbors. A remarkable advantage of the absence of shared communication medium is that we do not have to deal with potential network collisions/interferences.

A modular robot can be modeled by an undirected and unweighted graph of interconnected entities $G = (V, E)$ with V the set of vertices representing the modules, E the set of edges representing the connections, $|V| = n$, the number of vertices and $|E| = m$, the number of edges. We use the general concepts of graph theory such as the distance between two nodes and the diameter, d, of the graph.

In modular robotic systems where modules use neighbor-to-neighbor communications, the module spatial arrangement directly reflects the connectivity graph. Moreover, these systems often have a bounded number of connectors, i.e., of potential neighbors. As a direct consequence, large-scale ensembles tend to exhibit large hop distances. In (Naz et al., 2016c), we show that when arranged in a 3D lattice, the typical network distance between two random nodes is $O(\sqrt[3]{n})$ and the diameter is lower bounded by $\Omega(\sqrt[3]{n})$. This property should be considered in the design of synchronization protocol because, as explained in Section 2.5, the achievable synchronization precision is function of the hop distance.

Furthermore, our protocol is intended to synchronize fairly stable systems where changes in the network topology, due for instance to module mobility, or potential module or link failures, are infrequent.

3.2. Clocks: notation and assumptions

Each module M_i is equipped with its own internal clock and has its own local time $L^{M_i}(t)$, an approximation of the real time t. The goal of MRTP is to maintain a global timescale $G(t)$ across the system. We denote $G^{M_i}(t)$ module M_i's estimation of $G(t)$. MRTP preserves time monotonicity and prevents time from running backward, i.e., for any module M_i, $\forall(t, t'), t \geq t', G^{M_i}(t) \geq G^{M_i}(t')$. Moreover, we consider clocks which have high short-term frequency stability but which can be low precision and can have high skew relative to one another. Such clocks tend to drift apart from each other in a quasi-linear fashion over a short period of time.

We consider two synchronization error metrics. We define module M_i's relative synchronization error to the global time at real time t as:

$$\epsilon^{M_i}(t) = G^{M_i}(t) - G(t) \tag{1}$$

We define the maximum pairwise synchronization error at real time t, $\epsilon(t)$, as the maximum difference between any two global clocks in the system:

$$\epsilon(t) = \max_{M_i, M_j} \left| G^{M_i}(t) - G^{M_j}(t) \right| \quad (2)$$

Since our goal is to achieve global synchronization, we do not consider local-synchronization error metrics such as the maximum pairwise synchronization error between neighboring nodes (Lenzen et al., 2009).

3.3. Assumptions about network delays

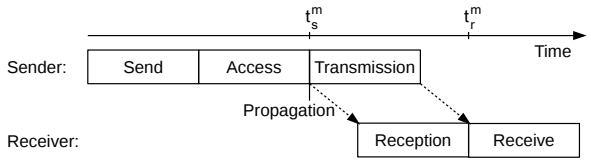

Figure 1: Sources of delivery delays in the exchange of a message m between two neighbor modules.

As indicated in (Ganeriwal et al., 2003; Maróti et al., 2004; Amundson et al., 2008), the exchange of a single message m between two neighbor modules can typically be characterized by the steps presented in Figure 1. Sending and receiving times represent the times for the message to travel from the application to the data-link layers. These delays are introduced by the operating system and are highly non-deterministic. The access time represents the waiting time at the data-link layer for accessing the communication channel. This time is also highly non-deterministic. The transmission and reception times represent respectively the times to transmit and to receive the frame using a bit by bit transmission at the physical layer. These delays are mainly deterministic and depend on the length of the frame and the bit rate. The propagation time represents the time for the bits to travel from the sender to the receiver over the physical link. This delay is highly deterministic and depends on the distance between the modules involved in the communication and on the propagation speed over the physical link. We define the transfer time, $T^m_{transfer}$, as the sum of the transmission, propagation and reception times for a message m. These times are highly deterministic.

3.4. Predictive Method to Compensate for Communication Delays

We propose to use the predictive method (PRED) to compensate for communication delays whenever they can be predicted. PRED is a naive method that relies on the assumption that $T^m_{transfer}$ is predictable with a certain accuracy that directly impacts the precision of our protocol. Moreover, it assumes messages can be timestamped at the data-link layer, shortly before the beginning of the transmission at time t^m_s and upon complete reception at time t^m_r. If we neglect interrupt handling time, $T^m_{transfer} = t^m_r - t^m_s$.

To compensate for communication delays, the predictive method (PRED) works as following. Let us assume that a module M_i receives a message m from a module M_j and that m was timestamped at the data-link layer on both sides (i.e., m contains $L^{M_j}(t^m_s)$ and $L^{M_i}(t^m_r)$). Then, the module M_i can compensate for communication delays of m and estimate the local time of M_j at the reception of m by:

$$L^{M_j}(t^m_r) \approx L^{M_j}(t^m_s) + T^m_{transfer} \quad (3)$$

4. The Modular Robot Time Protocol (MRTP)

MRTP works in two steps: The first step initializes the system: election of a central module as the time master *TM*, construction of a spanning-tree and initialization of the global clock. In the second step, the time master periodically synchronizes the slave modules.

4.1. Method to compensate for communication delays

The method to compensate for communication delays in MRTP has to be carefully selected depending on the target system. The choice of the technique to use has a direct impact on both the synchronization precision and the communication efficiency of MRTP. The precision of an approach mainly depends on the clock precision, its resolution and the communication mechanism. In Section 6.1.3, we describe a procedure to experimentally evaluate the precision of a given approach over multiple hops.

In that section, we also show that, in our target systems, the Blinky Blocks, PRED is in average more precise than the two other existing methods that can be applied to this system (i.e., FD and RTT). Moreover, PRED uses an unidirectional message exchange while RTT requires a bidirectional

message exchanges, thus incurring a larger communication overhead. In the rest of this section, we describe MRTP assuming PRED is used. Note that in practice, any method compatible with the target system can be used.

4.2. Step 1: Initialization

4.2.1. Time master election

A module is elected as the time master using an external algorithm. Different criteria can be used for the election of the time master (e.g., minimum-identifier node, etc.). As explained in Section 3.1, modular robotic systems with neighbor-to-neighbor communications form large-diameter networks. To achieve a better synchronization precision, we recommend to elect a central module as the time master, i.e., a node that tends to minimize the maximum or the average hop distance to any other module. Placing the time master close to the center of the system reduces the time of the synchronization phases and increases the overall precision, because cumulative estimations are made every hop. Any center election algorithm can be used. We suggest to use ABC-Center (Naz et al., 2015b), E2ACE (Naz et al., 2016a) or the algorithm presented in (Kim and Wu, 2013). These algorithms scale well in terms of memory usage and execution time. In our experimental evaluation, we use a modified version of E2ACE (see Section 6.2.2). Note that these three algorithms require every module has a unique identifier.

To handle dynamic topology changes, a module launches a time master re-election if it detects a new neighbor or a neighbor departure, and the system goes through the whole initialization process again. The technique based on an election index proposed in (Vasudevan et al., 2004) can be used for this purpose.

4.2.2. Breadth-first spanning-tree construction

At the end of the election process, our protocol creates a breadth-first spanning-tree rooted at the time master. The algorithm presented in (Cheung, 1983) combined with the optimizations described in Section 6.2.2 can be used. This algorithm guarantees that modules at distance d_{TM} hops of the time master in the physical configuration, are at distance d_{TM} hops in the tree. Logical neighbors in the tree are neighbors in the physical configuration. At this point, every module knows its parent and children in the tree. This tree will be used to recursively propagate synchronization waves from the time master through the system. As explained in Section 2.2, this approach is, in systems running under our assumptions, more communication efficient than infrastructure-less network-wide flooding based approaches.

4.2.3. Global clock initialization

Slave modules. Initially, slave modules estimate the global time with their local time. Slave modules adjust their estimation of the global time during synchronization phases, in the second step of MRTP. When a new time master is elected, modules keep their previous estimation of the global time but do not keep previous corrections of the clock skew. They can indeed disturb the synchronization process when two distinct systems are merged together.

Time master. Since time cannot run backward, clocks in advance of the global timescale have to slow down or to wait during synchronization process, and clocks behind the global timescale have to jump to it. To make time synchronization convergence faster, the global time is initially set to an estimation of the most advanced global time in the system using the convergecast-time-max algorithm inspired by (Raynal, 2013). This approach can cause important jumps into the future.

The pseudo-code of convergecast-time-max for any module M_i is provided in Algorithm 1. At any time, a module M_i estimates the maximum global time with:

$$Y^{M_i}(t) = L^{M_i}(t) + offset^{M_i}(t) \qquad (4)$$

With $offset^{M_i}(t)$ the estimated offset between M_i's estimation of the maximum global time in the system and M_i's local clock at time t. Initially, M_i considers it has the maximum global time (line 2). This algorithm uses a single type of message, namely *BACK* message. Every *BACK* message m is timestamped twice at the data-link layer: the sender M_j inserts $Y^{M_j}(t_s^m)$ just before transmission start and the receiver M_k inserts $L^{M_k}(t_r^m)$ upon complete reception (see Figure 1). Each leaf module sends a *BACK* message to its parent (line 6). Every non-leaf module waits for a *BACK* message from all its children. When M_i receives a $BACK(Y^{M_c}(t_s^m), L^{M_i}(t_r^m))$ message m from one of its children M_c, M_i estimates $Y^{M_c}(t_r^m) \approx Y^{M_c}(t_s^m) + T^m_{transfer}$ using the PRED

method (line 9) and adjusts $offset^{M_i}(t)$ accordingly (line 10).When M_i has received a $BACK$ message from all its children, it sends in turn a $BACK$ message to its parent. When the convergecast terminates (lines 4 or 13), the time master has an estimation of the maximum global time in the system $Y^{TM}(t)$. The time master then sets the global timescale $G(t)$ to $Y^{TM}(t)$. The convergecast-time-max algorithm neglects the effect of clock skew, and considers offsets to be constant in the system during convergecast.

Input: M_p // parent in the tree
$Children$ // set of children in the tree

1 **Initialization** of M_i at time t_{init}:
2 $offset \leftarrow G^{M_i}(t_{init}) - L^{M_i}(t_{init})$;
$Wait \leftarrow Children$;
3 **if** $M_p = \perp$ **then**
4 | // convergecast-max-time terminates
5 **else if** $Wait = \emptyset$ **then**
6 | send $m = \text{BACK}(\_,\_)$ to M_p;
| // M_p will receive BACK($Y^{M_i}(t_s^m) = L^{M_i}(t_s^m) + offset^{M_i}(t_s^m), L^{M_p}(t_r^m)$) at the application layer. $Y^{M_i}(t_s^m)$ is inserted by M_i at the data-link layer, just before transmission start. M_p will insert $L^{M_p}(t_r^m)$ upon reception, at the data-link layer.
7 **end**

8 **When** m = $\text{BACK}(Y^{M_c}(t_s^m), L^{M_i}(t_r^m))$ **is received by** M_i **from** M_c such that $M_c \in Children$ **do**:
9 $Y^{M_c}(t_r^m) \leftarrow Y^{M_c}(t_s^m) + T^m_{transfer}$;
10 $offset \leftarrow max(offset, Y^{M_c}(t_r^m) - L^{M_i}(t_r^m))$;
11 $Wait \leftarrow Wait - \{M_c\}$;
12 **if** $M_p = \perp$ **then**
13 | // convergecast-max-time terminates
14 **else if** $Wait = \emptyset$ **then**
15 | send $m' = \text{BACK}(\_,\_)$ to M_p;
| // As explain in comment line 6, M_p will receive BACK($Y^{M_i}(t_s^{m'}), L^{M_p}(t_r^{m'})$) at the application layer.
16 **end**

Algorithm 1: The convergecast-max-time algorithm for a module, M_i.

4.3. Step 2: Periodic synchronization

The time master holds the global timescale and periodically initiates synchronization phases. During each synchronization phase, the time master quickly disseminates the current global time along the edges of the breath-first spanning-tree built in the first step. $\tilde{G}(t)$, an estimation of the global time is recursively disseminated through the spanning-tree, module-by-module, starting from the time master. At each hop, the transmitted time is updated to take into account communication delays and time of residence in intermediate modules. Slave modules use a linear model to compensate for clock skew. As explained in the related-work section, this is a common choice.

The time master starts synchronization phases by sending all its children the current global time. Algorithm 2 details the synchronization process of any slave module M_i.

Input: M_p // parent in the tree
$Children$ // set of children in the tree
w // maximum number of synchronization points used for linear regressions

1 **Initialization** of M_i:
2 $a \leftarrow 1.0$; $b \leftarrow 0$; $W \leftarrow \emptyset$;

3 **When** m = $\text{SYNC}(\tilde{G}(t_s^m), L^{M_i}(t_r^m))$ **is received by** M_i **from** its parent M_p **do**:
4 $\tilde{G}(t_r^m) = \tilde{G}(t_s^m) + T^m_{transfer}$;
5 **if** $|W| = w$ **then**
6 | $W \leftarrow W - \{\underset{\tilde{G}(t)}{\text{argmin}}\ W(<\tilde{G}(t), L(t)>)\}$;
7 **end**
8 $W \leftarrow W \cup <\tilde{G}(t_r^m), L^{M_i}(t_r^m)>$;
9 computeLinearRegression(a, b, W);
10 **for each** $M_c \in Children$ **do**
11 | send $m' = \text{SYNC}(\_,\_)$ to M_c;
| // M_c will receive SYNC($\tilde{G}(t_s^{m'}), L^{M_c}(t_r^{m'})$) at the application layer. $\tilde{G}(t_s^{m'}) = \tilde{G}(t_r^m) + a^{M_i}(W^{M_i}(t_s^{m'})) * (L^{M_i}(t_s^{m'}) - L^{M_i}(t_r^m))$ is inserted at the data-link layer, just before transmission start. M_c will insert $L^{M_c}(t_r^{m'})$ upon reception, at the data-link layer.
12 **end**

Algorithm 2: Synchronization protocol for a slave module, M_i.

4.3.1. Time-stamping and Global Time Estimation

The synchronization process uses a single type of message $SYNC$. Every $SYNC$ message m is timestamped twice at the data-link layer: the sender, M_j, inserts $\tilde{G}(t_s^m)$ just before transmission start and the receiver, M_k, inserts $L^{M_k}(t_r^m)$ upon complete reception. When M_i receives a $SYNC(\tilde{G}(t_s^m), L^{M_i}(t_r^m))$ message m from its parent, M_i computes $\tilde{G}(t_r^m) = \tilde{G}(t_s^m) + T^m_{transfer}$, an estimation of the global time at the reception of the synchronization message, using the PRED method (line 4). $<\tilde{G}(t_r^m), L^{M_i}(t_r^m)>$ forms a synchronization point that contains both M_i's local

clock value and the estimation of the global time at nearly the same real time. M_i can estimate its relative synchronization error to the global time by $G^{M_i}(t_r^m) - \tilde{G}(t_r^m)$.

4.3.2. Global Clock Adjustment

M_i computes $a^{M_i}(W^{M_i}(t))$ and $b^{M_i}(W^{M_i}(t))$ such that

$$\tilde{G}(t) \sim a^{M_i}(W^{M_i}(t)) \times L^{M_i}(t) + b^{M_i}(W^{M_i}(t))$$

using least-squares linear regression based on $W^{M_i}(t)$, a window of the last w synchronization points (line 9). $a^{M_i}(W^{M_i}(t))$ denotes M_i's estimated skew relative to the global time, and $b^{M_i}(W^{M_i}(t))$ its estimated offset at time t. This mechanism compensates for clock skew and enables modules to be synchronized less frequently without degrading the synchronization precision. In order to preserve time monotonicity, our protocol prevents $G^{M_i}(t)$ from running backward:

$$\begin{aligned}&\forall(t,t'), t \geq t',\\ &G^{M_i}(t) = max((G^{M_i}(t'), \quad a^{M_i}(W^{M_i}(t)) \times L^{M_i}(t)\\ &\qquad\qquad + b^{M_i}(W^{M_i}(t)))\end{aligned}$$

If a new computed model leads to an estimated global time behind the maximum time already reached by $G^{M_i}(t)$, then $G^{M_i}(t)$ is blocked until the new model reaches this maximum time. Otherwise, $G^{M_i}(t)$ jumps into the future.

4.3.3. Global Time Dissemination

Shortly after reception of a synchronization message m, M_i sends a *SYNC* message m' to each of its children M_c in the tree (line 11). Hence, synchronization waves are quickly disseminated through the whole network leading to a better synchronization precision (Lenzen et al., 2009). At the data-link layer, M_i inserts

$$\begin{aligned}\tilde{G}(t_s^{m'}) = \tilde{G}(t_r^m) +\\ a^{M_i}\left(W^{M_i}\left(t_s^{m'}\right)\right) \times \left(L^{M_i}\left(t_s^{m'}\right) - L^{M_i}\left(t_r^m\right)\right)\end{aligned}$$

into m', just before it starts to transmit the frame over the communication medium. This compensates for the time of residence at module M_i, assuming M_i clock skew to be constant and equal to $a^{M_i}(W^{M_i}(t_s^{m'}))$ during this time. M_c inserts its local time $L^{M_c}(t_r^{m'})$ into the incoming message at the data-link layer, immediately after M_c pulls the synchronization message from the interface buffer. At M_c's application layer, m' contains $\tilde{G}(t_s^{m'})$ and $L^{M_c}(t_r^{m'})$. M_c then repeats the same synchronization process than M_i.

4.3.4. Synchronization Periods

Our protocol contains two synchronization phases: a calibration phase and a runtime phase. During the calibration phase, modules are more frequently synchronized with a period P_{ca} in order to collect enough synchronization points to compute skew models while preserving a satisfying level of precision. The calibration phase lasts $w \times P_{ca}$. Then, during the runtime phase, modules are synchronized less frequently, with a period P_{ru}, and use the computed models to compensate for clock skew. The values of w, P_{ca}, and P_{ru} have to be chosen according to the target platform hardware and the desired precision, with resource usage in mind. In our experimental evaluation, we empirically selected $w = 5$, $P_{ca} = 2$ *seconds* and $P_{ru} = 5$ *seconds* (unless otherwise mentioned). These values provide, in our target platform, a satisfactory precision at a reasonable cost in terms of communications and computations.

5. The Target System: the Blinky Blocks

We implemented MRTP and evaluated it on the Blinky Blocks system (see Figure 2) using both hardware prototypes and a simulator for modular robots called VisibleSim (Dhoutaut et al., 2013). This section presents the Blinky Blocks platform and the models we use to simulate this system.

5.1. General characteristics

Blinky Blocks are centimeter-size blocks that can be attached to each other using magnets. Each block is equipped with an ATMEL ATxmega256A3-AU 8/16-bits 32-MHz micro-controller having 256KB ROM and 16KB RAM (ATMEL, 2013). All the blocks of a system execute the same program. Blocks communicate with their neighbors through serial interfaces on their faces. A single block is connected to a power-supply. Power is distributed across the system using dedicated pins. Modules can change their color thanks to embedded RGB LEDs. Furthermore, a distributed logging system enables all modules to send information to a computer connected to the system using a serial connection.

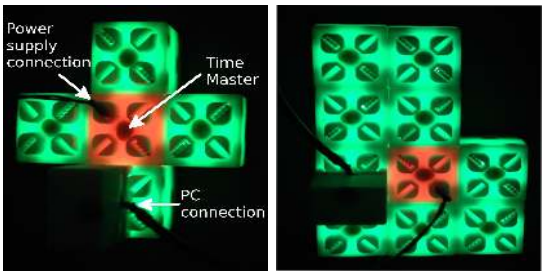

Figure 2: Two Blinky Blocks systems synchronized using MRTP. On the left, the system forms a cross. On the right, blocks are deployed in a doubled L-configuration. In both configurations, the time master, in red, is connected to the power supply. Slave modules are in green. Experiment data are sent by the systems to the PC through a serial cable.

5.2. Local clock properties

5.2.1. Hardware system

Each module maintains its local time using a Real-Time Counter (RTC) driven by an internal RC oscillator running at a frequency of 1.024 kHz with an accuracy of 1% (10,000 ppm), at 3V and 25°C (ATMEL, 2016). The RTC counts the time elapsed since the module started with a resolution of about 0.98 millisecond[5]. Thus, the synchronization precision results announced in the evaluation section are actually expressed in 0.98 a millisecond, even though we express them in milliseconds for sake of simplicity. It is important to understand that these oscillators exhibit very poor accuracy and low resolution that directly affects the performance of our protocol. For instance, a frequency deviation of 1% causes a clock error of approximately 10 milliseconds per second. Most previous work on time synchronization, e.g., (Elson et al., 2002; Ganeriwal et al., 2003; Maróti et al., 2004; Schenato and Fiorentin, 2011), were evaluated on devices equipped with crystal oscillators that have a typical accuracy between 0.0001% and 0.01% (1 to 100 ppm) and a resolution on the order of tens of microseconds. Under constant temperature and constant supply voltage, RC oscillators are fairly stable over a short period of time. As shown in Figure 3, Blinky Blocks local clocks tend to drift apart in a roughly linear fashion on the short-term.

5.2.2. Simulation model

In (Allan, 1987), the authors propose a general model for oscillators:

$$L^{M_i}(t) = \frac{1}{2}D^{M_i}t^2 + y_0^{M_i}t + x_0^{M_i} + \eta^{M_i}(t) \qquad (5)$$

where t is the real time (i.e., simulation time), $L(t)$ is the local time, x_0 is the time offset, y_0 is the frequency offset, D is the frequency drift and $\eta(t)$ is a random noise. As explained in (Allan, 1987), y_0 and D may vary over time (e.g., because of aging, temperature variations, etc.). For the sake of simplicity, we consider them to be constant and express their small variations in the noise signal $\eta(t)$.

We assume that Blinky Blocks clocks follow (5)'s model. We conducted experiments on hardware using Blinky Blocks in order to compute model parameters. We used a system of five blocks deployed in a cross configuration (see Figure 2) to collect time reference points $< t, L^{M_i}(t) >$, with i the block unique identifier, every 10 seconds during 7 hours (see Figure 3). The real time t was provided by a computer. We assumed the computer clock to be perfect. We use the PRED method to compensate for communication delays.

Figure 4 shows the distribution of the parameter values obtained using polynomial regression with R. The parameters D and y_0 seem normally distributed. As a consequence, we randomly generate clock parameters following normal distributions with the corresponding mean and standard deviation (see Table 2). Noise signals are the residual standard errors. We extracted the 5 noise signals and replay them in our simulations.

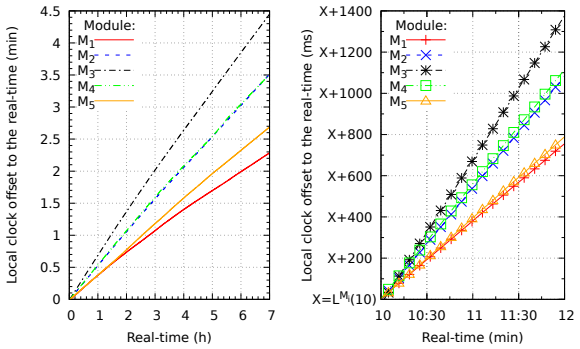

Figure 3: Local lock offset to the real-time ($L^{M_i}(t) - t$). The plot on the left shows the long term deviation of the local clocks, while the plot on the right shows these deviations on a shorter term.

[5]Resolution= $\frac{1}{1.024} \approx 0.98ms$

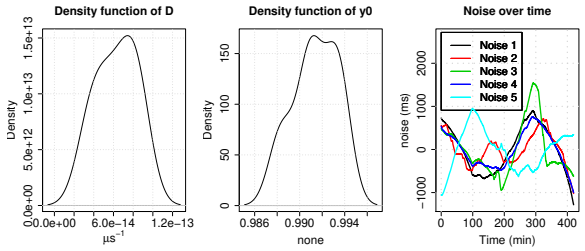

Figure 4: From the left to the right: D density function, $y0$ density function and the noise signals over the time.

Parameters	Simulation Model
D (μs^{-1})	$\mathcal{N}(7.132315 \cdot 10^{-14}, 5.349995 \cdot 10^{-14})$
y_0 $(none)$	$\mathcal{N}(0.9911011, 0.002114563)$
x_0 (μs)	additive inverse of the simulation time at module startup
η (μs)	Noise replayed from extracted data signals

Table 2: Blinky Blocks hardware-clock model parameters used in VisibleSim. $\mathcal{N}(\mu, \sigma)$ refers to the normal probabilistic law with μ mean and σ standard-deviation.

5.3. Communication properties

5.3.1. Hardware system

Blinky Blocks use full-duplex neighbor-to-neighbor communications over serial links controlled by Universal Asynchronous Receivers/Transmitters (UARTs) configured with a bit rate of 38.4 kbauds. Modules exchange messages that contains up to 17 bytes of data.

A message is sent over the link into a frame composed of 21 bytes minimum: 17 bytes of payload data and 4 bytes of control (i.e., frame delimiter, checksum, etc.). Some special bytes need to be escaped using an extra byte in order to dissociate command bytes from data ones. Thus, the number of bytes actually sent on the link varies a little bit according to the data being sent.

A frame is transfered byte per byte to/from the UART. The transfer is interrupt controlled, i.e., the UART generates an interrupt when it has finished transmitting or receiving a byte. The transmission time starts when the first byte of data is moved to the UART buffer and ends when the last byte leaves this buffer. The reception time starts when the first byte of data is received by the UART and ends when the last byte is received.

5.3.2. Transfer time estimation

The PRED method used to compensate for communication delays in MRTP assumes the transfer time, defined in Section 3.3, to be predictable. The transfer time includes the transmission time, the propagation time and the reception time. The Blinky Blocks are identical and physically connected, thus the propagation time between two neighbor modules can be considered as deterministic. The transmission time and the reception time of a message depends on the actual frame size and the communication rate.

$T_{transfer}$ can be estimated using two-way timestamped-message exchanges (see Figure 5 and Equation (6)). Equation (6) assumes communication delays for frames of same size to be symmetrical. In addition, the exchange of messages is assumed to be fast enough so that the skew between the two module's clocks is insignificant during the exchange.

$$T_{transfer} \approx \frac{(L^{M_2}(t_r^{m\prime}) - L^{M_2}(t_s^m)) - (L^{M_1}(t_s^{m\prime}) - L^{M_1}(t_r^m))}{2} \tag{6}$$

We experimentally measured $\widetilde{T}_{transfer}$ for 300,000 two-way message exchanges between neighbor modules in sparse and compact Blinky Blocks systems (see Figure 5). We observed that $\widetilde{T}_{transfer}$ is always between 5 and 7 milliseconds. On average, $\widetilde{T}_{transfer}$ of 21-byte long frames varies slightly around 6 milliseconds depending on the number of simultaneous communications. Moreover, at the resolution of 1 millisecond, the transfer time of identical-length frame is fairly constant. A transfer time of 6 milliseconds for a 21-byte long frame corresponds to a transfer rate of 28 kbit/s. Based on these results, we consider that the transfer rate of a message can be estimated by $\widetilde{R}_{transfer} = 28kbit/s$. As a consequence, we use Equation (7) to estimate the transfer delay of a message and to compensate for communication delays in the PRED method.

$$\widetilde{T}_{transfer} = \frac{\text{frame size}}{\widetilde{R}_{transfer}} \tag{7}$$

5.3.3. Simulation model

In order to accurately simulate the time, our simulation model takes into account the timeout triggering time, the processing times, the queueing delays, and the transfer rate of the messages (see Figure 6). We did not observe any node crash nor any transmission failure or message loss during the

experiments of the previous section, when the network is not overwhelmed. Thus, our simulation model does not incorporate any special mechanism to mimic such phenomena. Table 3 summarizes the different random variables of our model.

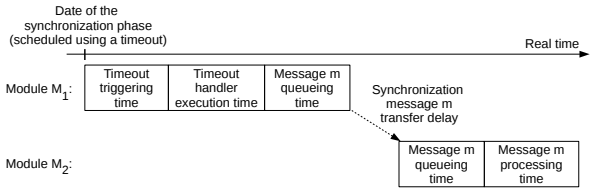

Figure 6: Communication model workflow through an example. Module M_1 has scheduled a synchronization phase. Upon timeout expiration, module M_1 executes the synchronization procedure and sends a synchronization message to module M_2 which will process it after a possible delay due to queueing.

Timeout triggering time. The timeout triggering time is the amount of time a module needs to trigger an action scheduled using a software timeout (e.g., the synchronization timeout that initiates a synchronization phase). In the Blinky Blocks firmware, software timeouts are checked with a frequency of 2000 Hz. Thus, if we neglect the interrupt delay, an action scheduled at time t can be executed at any time t' such that $t \leq t' \leq t + 500\mu s$.

Processing time. We use the micro-controller clock running at 32 MHz (nano-second scale resolution) to measure the processing time of the synchronization-timeout handler and the synchronization-message handler. We define two generic models to simulate the message handler processing time: one for handlers with low-computation cost (e.g., clock adjustment without linear regression) and one for handlers with medium-computation cost (e.g., clock adjustment with linear regression computation on a window of 5 measures).

Note that the queueing and transfer delays include some processing time. In our evaluation, modules were running a rather simple application, in which every module periodically changes its color based on the current global time and does nothing the rest of the time using an active sleep (while loop with a time limit). Thus, they were actually computing all of the time. The transfer time includes the interrupt time to fetch bytes from the interface buffer. Our target platform (and many others) uses interrupt driven communications. Hence, only a very few elementary micro-controller instructions are executed before a byte is fetched. We reasonably assume that interrupts are never disabled and that there are not a large number of interrupts to be simultaneously handled. The queueing delays include interrupt time to enter the routine that handles incoming messages and the time to handle potential messages that were already present in the queue at the message arrival.

Queueing delays and network load. VisibleSim uses a queueing system to handle both incoming and outgoing messages. We propose two queue load models. The first model is dedicated to lightly loaded networks where modules only exchange neighborhood management messages, with a period of 500 milliseconds. The second model is intended to simulate moderate network traffic due to extra-applications running on the nodes. In this model, in addition to simulate the neighborhood management messages, the queue occupancy at a message arrival follows a Poisson distribution of mean 1. This simulates a moderate network traffic in which message queues contain most of the time 0 to 2 messages and in few cases more messages. The light-load model is used in the experiments of sections 6.1. The moderate-load model is used in our evaluation on large-scale systems (see Section 6.2).

Transfer rate. Below the millisecond unit, the transfer rate is scenario dependent. It depends, for instance, on the number of simultaneous communications. For each experiment performed on the hardware platform, we empirically derive the average system transfer rate using statistics on the round-trip time. We use similar experiments to the ones presented in Section 6.1.3. We define three transfer rate models, namely for sparse, intermediate and compact systems. In a given simulation, all the modules use the same transfer rate model. The model for sparse systems is used in the experiments of Section 6.1.3, on the line system. The model for intermediate systems is used in the experiment of Section 6.1.4, on the L-shaped system (see Figure 2). The model for compact systems is used in our evaluation on large-scale systems (see Section 6.2).

6. Evaluation

This section presents our experimental evaluation of MRTP, performed both on hardware Blinky Blocks and in the VisibleSim simulator. Through our experiments, we show the effectiveness, the efficiency and the scalability of our protocol. More precisely, we first evaluate the precision of MRTP on hardware and show through some examples that VisibleSim accurately simulates Blinky Blocks systems. Then, we use VisibleSim to evaluate the performance of MRTP in large-scale systems and to compare it to existing synchronization protocols in terms of precision, time of convergence and communication efficiency.

6.1. Evaluation on hardware and simulation fidelity

In this section, we evaluate the precision of the synchronization achieved by MRTP on the Blinky Blocks hardware. In addition, we show that VisibleSim accurately simulates Blinky Blocks systems.

6.1.1. Methodology

We first use color changes to show that MRTP can potentially manage systems composed of up to $27,775$ Blinky Blocks.

Then, we show how the hop distance impacts the precision of the estimated global time, $\tilde{G}(t)$, disseminated through the network during synchronization phases. We compare different methods to compensate for communication delays and show that the PRED method (see Section 3.4) is in average the most accurate in our target platform. Furthermore, we show that within a few hops, $\tilde{G}(t)$ can be used as a reference time to estimate the relative synchronization error of the Blinky Blocks to the global time.

We then use this estimation to study the local clock behaviors and to show the impact of various parameters on the precision of our protocol.

All experiments presented in this section were one-hour long. Unless otherwise mentioned, modules were synchronized every 2 seconds in the calibration phase, then every 5 seconds in the runtime phase and modules used five synchronization points for the linear regressions. These values were empirically chosen with the aim of obtaining, a satisfactory synchronization precision in practice, at reasonable computation and communication costs. Moreover, unless otherwise indicated, the PRED method is used to compensate for communication delays.

6.1.2. Evaluation of the precision of MRTP using color changes

Measuring the synchronization precision using message exchanges is as challenging as performing time synchronization since both consist in measuring clock offsets. It is difficult, mainly because time keeps going during communications.

In this subsection, we apply MRTP on a system of 28 Blinky Blocks that have to simultaneously change their color. Potential delays between module color changes reflect the synchronization error of the modules. Modules are connected in a line topology. The time master is manually placed on an extremity of the system and it synchronizes the other modules every 500 milliseconds. With a such runtime synchronization period, every link of the synchronization tree is theoretically used by MRTP only about 1.2% of the time[6]. Slave modules have to simultaneously change their color every 3 seconds. This experiment was recorded using a 40 millisecond resolution camera.

We observed that every time the system starts to change its color, all slave modules have changed their color on the next image, 40 milliseconds latter (see Figure 7). Hence, MRTP is potentially able to synchronize a system with a radius of up to 27 hops to less than 40 milliseconds, if the time master is at the center of that system. To give an order of magnitude, a Blinky Blocks system with a radius of 27 hops can be composed of up to $27,775$ modules and have a diameter of 54 hops, as demonstrated in (Naz et al., 2016c).

In the next subsections, we present a more precise and automatized evaluation of MRTP.

6.1.3. Impacts of the hop distance on the precision of the disseminated global time $\tilde{G}(t)$ for different compensation delay methods

We expect that the estimation of the global time disseminated during synchronization phases, $\tilde{G}(t)$, gets less precise as the depth of the synchronization tree increases because small but cumulative errors in the estimations of the global time are made at every hop. In this section, we first propose a generic method to evaluate compensation delay methods over multiple hops. Then, we present results obtained using the FD, RTT and PRED methods

[6] $\frac{T_{transfer}}{P_{ru}} \approx \frac{6}{500} \approx 1.2\%$ (without retransmission due to potential message loss or corruption)

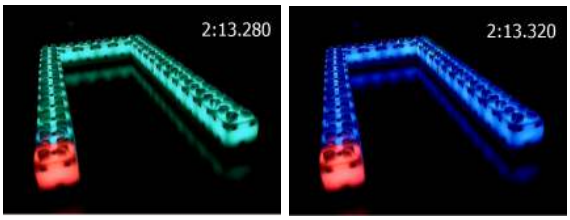

Figure 7: Two successive images of a video recording 28 Blinky Blocks connected in a line topology and synchronized using MRTP. The time master is in red and intentionally placed on an extremity of the line. Slave modules have to simultaneously change their color every 3 seconds. On the left, a color change starts in the system. On the right, 40 milliseconds after, the color of every slave module has changed.

to compensate for communication delays (see Sections 2.3 and 3.4). We show that the PRED method is in average more accurate. Finally, we show that within a few hops, $\tilde{G}(t)$ can be used as a reference time to estimate the synchronization error of the Blinky Blocks.

Methodology. We evaluate the precision of $\tilde{G}(t)$ using virtual modules emulated on Blinky Blocks hardware systems. Figure 8 gives the intuition behind our experiments in a line system. This method, inspired by the approach presented in (Römer et al., 2005), allows us to compare the estimated global time received by the module M_{2n-1} to the actual global time held by the time master $TM = M_1$, because these two modules are emulated on the same physical block and can both read the actual global time $G(t)$.

In the example depicted in Figure 8, every physical block hosts 2 virtual modules except one block. Each slave virtual module maintains its own estimation of the global time. The synchronization tree rooted at the time master TM links the virtual modules together in a virtual line, such that neighbor modules in the tree are hosted on a separate physical block. The leaf module M_{2n-1} is at a distance of $2(n-1)$ hops from TM in the synchronization tree. M_{2n-1} computes the global time dissemination error as $G(t) - \tilde{G}(t)$.

In our experiments, we generalize the example of the virtual line to measure the global-time dissemination error versus the hop distance in arbitrary systems. Modules host a number of virtual modules equal to the diameter of the system, and each physical module initiates a return trip to the root of the tree. The root of the tree receives timing messages that have physically traveled from 2 hops to $2(d-1)$ hops (or $2d-1$, if the diameter is odd).

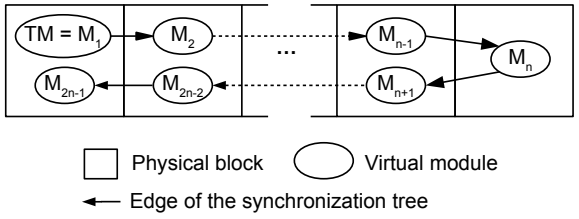

Figure 8: Scheme of a virtual line of emulated modules on hardware Blinky Blocks connected in a line.

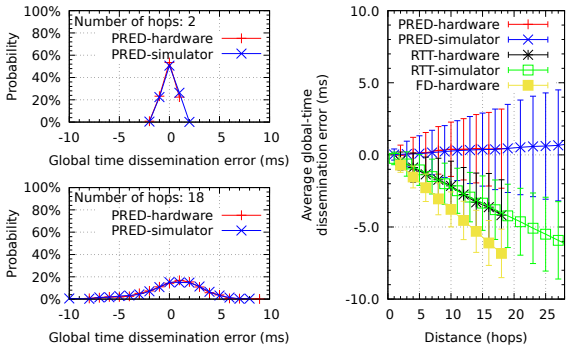

Figure 9: Dissemination error (± standard-deviation) according to the hop distance using different methods to compensate for communication delays. On the left, the distribution of the error. On the right, the average error (± standard deviation).

Results. Figure 9 and Table 4 show the impact of the hop distance on the global time dissemination error. As announced in Section 2.5, the absolute mean error increases linearly with the number of hops and the standard-error tends to increase with the square-root of the number of hops. As a consequence, placing the time master at the center of the system appears as a judicious choice.

It appears that PRED is in average more precise than FD and RTT methods, both in sparse and more compact systems. This is mainly due to the fact that, in Blinky Blocks systems, the average transfer delay of a frame is almost a round number at the millisecond scale. We observe that regardless of the distance, the error distribution of PRED seems Gaussian and nearly centered around zero.

Note that PRED has a more important standard-deviation than the two other methods. FD has

the smallest standard-deviation as only the transfer time of a single byte is involved in the estimation of the global time whereas PRED and RTT use the transfer time of complete messages.

For a distance of 4 hops, 95% of the error measures are between [-2;2] milliseconds and the average error is close to zero. Because of the poor accuracy of the Blinky Blocks hardware clocks, we expect synchronization error using our protocol to be greater than 1 to 2 milliseconds. Thus, within a few hops, $\tilde{G}(t)$ can be used as a reference time to estimate the synchronization error of the Blinky Blocks. Upon reception of a synchronization message, a module M_i estimates its relative synchronization error to the global time by $\tilde{\epsilon}^{M_i}(t) = G^{M_i}(t) - \tilde{G}(t)$. We do not use virtual modules any more in the rest of the evaluation.

Compensation delay method	Average global time dissemination error ± standard-deviation (ms)			
	Line configuration		Compact configuration	
	2 hops	4 hops	2 hops	4 hops
PRED	-0.03 ± 0.70	-0.11 ± 1.11	-0.27 ± 0.67	-0.36 ± 1.02
RTT	-0.42 ± 0.62	-0.88 ± 1.01	-0.50 ± 0.63	-0.80 ± 0.97
FD	-0.71 ± 0.50	-1.53 ± 0.76	-0.87 ± 0.54	-1.63 ± 0.80

Table 4: Average dissemination error (± standard-deviation) to the global time for 2 and 4 hops using the different methods to compensate for communication delays in both the line and the compact systems.

6.1.4. *Impact of the synchronization periods on the synchronization precision*

Figure 10 shows the impact of the synchronization periods on the relative synchronization error in the doubled L-shaped system depicted on Figure 2. Distributions seem Gaussian. They are all bell-shaped and centered around 0. For a runtime synchronization period of 5 seconds, the average relative synchronization error is equal to 0.22 milliseconds.

We observe in Figure 10 that the distribution shape becomes shorter and larger as the runtime synchronization period increases. The error dispersion reflects the synchronization error. The standard deviation increases with the runtime synchronization period. As a consequence, the longer the resynchronization interval is, the worse the synchronization precision will be. However, it must

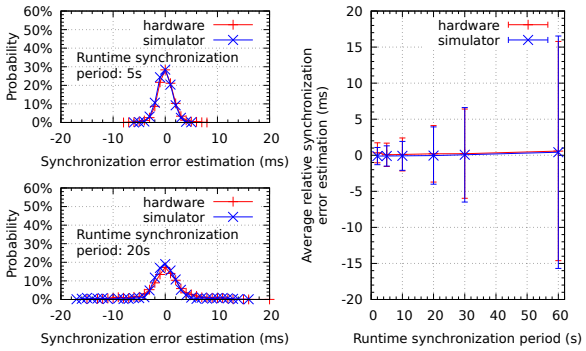

Figure 10: Relative synchronization error of the whole system according to the synchronization periods. The system is synchronized using MRTP. On the left, the distribution of the error. On the right, the average error (± standard deviation).

be noted that in all cases, the system stays synchronized to a few milliseconds. The average synchronization error amplitude remains below 4 milliseconds for runtime synchronization periods ranging from 2 seconds to 30 seconds. With a runtime period of 5 seconds, every link of the synchronization tree is theoretically used by MRTP only about 0.12% of the time during the runtime phase.

6.1.5. *Impact of the number of synchronization points used for the linear regressions on the synchronization precision*

Figure 11 shows the impact of the number of synchronization points used for the linear regressions on the synchronization error in the doubled L-shaped system depicted on Figure 2. With a running synchronization period of 5 seconds, we observe the maximum synchronization precision is obtained using 5 synchronization points for the linear regressions. Indeed, when using 5 synchronization points, the relative synchronization error has a mean close to 0 and the smallest standard-deviation. We suppose that, when using less points, the clock models are worse captured. When using more than five synchronization points, the synchronization precision decreases as the window size increases. We believe, without proving it, that this is because the clock frequencies vary too quickly for a large number of observations.

6.1.6. *Simulation fidelity*

As shown in Figures 9 and 10, results obtained using simulations closely matched the results from

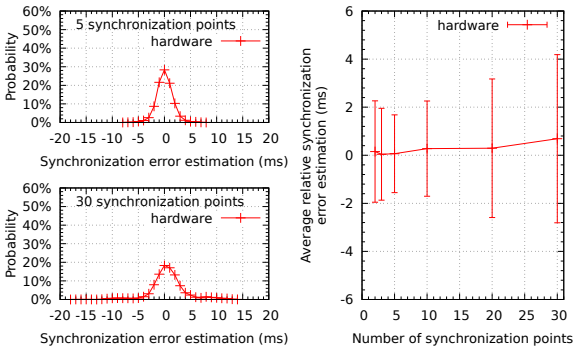

Figure 11: Relative synchronization error of the whole system according to the number of synchronization points used for the linear regressions. The system is synchronized using MRTP. On the left, the distribution of the error. On the right, the average error (± standard deviation).

the hardware-based experiments. Indeed, the results of the simulations almost have identical distribution and statistical measure values (i.e., mean and standard-deviation) than the results of the experiments on hardware systems. The global time dissemination error according to the hop distance is well simulated even after many hops. Thus, we can safely assume that VisibleSim can be used to study the performance of synchronization protocols in large-scale Blinky Blocks systems.

Note that we do not simulate the experiment in Section 6.1.5 because we did not compute processing times for linear regression on a large number of synchronization points (i.e., more than five) in our simulation model.

6.2. Large-scale evaluation and comparisons to existing protocols through simulations

In this section, we use the VisibleSim simulator to evaluate the performance of our protocol and compare it with existing synchronization protocols. We first present our methodology and experimental scenario. Then, we describe the variants of the synchronization protocols we use for comparisons. Finally, we show and discuss our simulation results.

6.2.1. Methodology and scenario

We study the precision, the convergence time and the communication efficiency of the synchronization protocols on three systems of different size and diameter (see Table 5). These systems are organized in a ball topology, i.e., the largest network topology that can be formed for a given diameter (Naz et al., 2016c). We use this compact network topology because there is an increasing number of modules, thus an increasing number of clock models, at a given network distance from any given module. Moreover, we consider the ball system composed of 27,775 modules to show that MRTP can effectively synchronize this system to a few milliseconds as announced in Section 6.1.2.

To compare protocols fairly, we evaluate them on identical systems, i.e., for all experiments, a module always has the same position, the same communication model and the same clock parameters. In addition, for centralized protocols, the time master always has the same communication model and clock parameters. Furthermore, the minimum-identifier module is deliberately placed on the extremity of the systems in order to show the impact of the maximum hop distance to the time master on the overall synchronization precision of the system.

All the experiments last two hours. During the first hour, the system is left unsynchronized. Then, modules starts running one of the considered synchronization protocol. For all the protocols, we use a synchronization period of 5 seconds. In protocols that use a linear model to compensate clock skew, modules performs the model parameter estimations using the last 5 synchronization points, unless otherwise mentioned. To evaluate the synchronization precision, we measure the maximum pairwise synchronization error, every 3 seconds.

System	Size (modules)	Radius (hops)	Diameter (hops)
Ball(5)	231	5	10
Ball(15)	4,991	15	30
Ball(27)	27,775	27	54

Table 5: Network characteristics of the systems used for the evaluation.

6.2.2. Compared synchronization protocols and modifications

We compare MRTP to leading protocols designed for ad-hoc networks, namely MLE_TPSN (i.e., TPSN (Ganeriwal et al., 2003) combined with MLE (Leng and Wu, 2010)), FTSP (Maróti et al., 2004), PulseSync (Lenzen et al., 2009), WMTS (He et al., 2014a) (a variant of MTS (He et al., 2014a)) and ATS (Schenato and Fiorentin, 2011). These protocols were proposed for wireless sensor networks and need modifications to be used on our

target platform. This section lists these modifications. Note that the operated modifications do not alter the general high-level framework of the compared protocols.

Communication medium. One of the adaptation is to consider a local and wired communication medium instead of a wireless and shared one. The main differences this adaptation causes from a data-link point of view are twofold. First, it entails the absence of message loss due to interferences/collisions on the communication medium. Second, in order to broadcast a message to all neighbors, a node has to send an individual copy of that message to all of them.

Communication delay compensation. As explained in the related work section, the methods used by these protocols to compensate for communication delays are not all directly applicable to our target platform. We recall that three methods are applicable to our target system, namely RTT, FD and PRED (see Section 2.3). MLE_TPSN uses round-trip messages and computes the maximum likelihood estimation of the current global time on the last 5 synchronization points. We use FTSP with PRED because the method proposed in FTSP, which is highly accurate, is not applicable to our target system and because PRED is in average the most precise method for our system. PulseSync employs the same method than FTSP, thus we use PulseSync with PRED. In ATS, the authors suggest to use the most precise method and utilize FD for the experimental evaluation. Since PRED is in average more precise than FD in our system, we use ATS with PRED. We also use PRED to compensate for communication delays in WMTS.

The ATS and the WMTS protocols. ATS and WMTS are respectively average and maximum-value consensus-based decentralized protocols. WMTS and ATS compensates for clock skew using averaging techniques. In WMTS and in the original version of ATS, modules use the two last clock readings of a neighbor to estimate its relative clock skew. In our modified version of ATS, we use the oldest and the newest clock readings to estimate the relative clock skew. This modification leads to better performance in our system. The ATS protocol takes input parameters, e.g., the probability to update the clock offset and the clock skew of the modules at each synchronization round. We use the parameters used in the evaluation section of the original article (Schenato and Fiorentin, 2011).

The MRTP and the TPSN protocols. MRTP and TPSN are centralized protocols in which modules get periodically synchronized with the time master. In MRTP and TPSN, the time master is elected using an external algorithm and child modules are recursively synchronized by their parents along the edges of a spanning-tree. For the leader election problem, we consider the minimum-identifier leader election algorithm (Raynal, 2013) (MIN_ID for short) and the E2ACE centrality-based election algorithm (Naz et al., 2016a). We use the algorithm presented in (Cheung, 1983) to build the synchronization tree. We slightly modified these three algorithms to reduce the number of messages sent and the maximum message queue length. In these algorithms, modules flood the best known solution to reach a consensus. In our modified version of these algorithms, a module does not send to a neighbor a better solution until it gets an acknowledgment message from its neighbor that confirms the previous sent solution was properly handled. Moreover, our version of E2ACE uses the HyperLogLog probabilistic counter (Flajolet et al., 2007), which is more efficient than the counter used in the original version of E2ACE. In addition, we slightly adapt the formula to elect an approximate-center node rather than an approximate-centroid one.

In (Ganeriwal et al., 2003), the author states that any method can be used to select the time master in TPSN and suggests that the minimum-identifier election algorithm presented in (Malpani et al., 2000) can be used. Thus, we use TPSN with MIN-ID. In addition, in the original version of TPSN, child modules overhear the messages exchanged during the synchronization process of their parent. As our platform uses contact communications, messages sent to a node can not be overheard by other nodes. Thus, in our version of TPSN, we added an extra message sent by the parent to trigger the synchronization of child modules. Moreover, modules use a linear model and MLE (Leng and Wu, 2010) to estimate the clock parameters. During a synchronization phase, modules only use the last-timing information to disseminate the global time through the system. Without this last modification, MLE_TPSN diverges slowly in our simulations.

The FTSP and the PulseSync protcols. FTSP and PulseSync are centralized protocols in which modules get periodically synchronized with the time master. FTSP and PulseSync are infrastructureless. During the synchronization phases, the minimum-identifier module gets implicitly elected as the time master. If a module has not received new synchronization messages for some synchronization periods (5 in our implementation), it declares itself as the time master and starts synchronizing the other modules. A module updates its belief about the current time master in the system whenever it receives a synchronization message advertising for a time master with a lower identifier.

In FTSP, a new time master ignores synchronization messages advertising for lower-identifier nodes during 3 synchronization periods. The FTSP protocol also takes as a parameter the number of synchronization messages a node needs to have received before it considers itself as synchronized and starts to synchronize neighboring nodes. In our simulations, we use the value of 3. For a better performance, we proceed to the subsequent modifications of FTSP, also suggested in (Lenzen et al., 2009). In the original version of FTSP, synchronized modules ignore the received global time values that are too far from their own estimation of the current global time. As shown in the next subsection, FTSP does not provide precise synchronization in our target system and we had to suppress this filtering procedure in order to obtain better results. Additionally, in our version of FTSP, modules clear their linear regression table whenever they get synchronized by a new time master.

PulseSync accurately synchronizes nodes using rapid network-wide flooding. Sophisticated methods have been proposed to achieve fast flooding in WSN where messages may interfere and collide with each others (e.g., (Ferrari et al., 2011)). Our target system does not assume any specific mechanism to quickly disseminate a message through the network. Blinky Blocks networks are beside not prone to message collisions. In our implementation of PulseSync, synchronization messages are handled like any other message. In particular, messages are not prioritized in message queues.

Naming convention. We use the following format to name the different compared approaches: [ORIGINAL PROTOCOL NAME]-[LEADER ELECTION ALGORITHM]-[COMMUNICATION DELAY COMPENSATION METHOD]. For instance, MRTP-EA2CE-PRED refers to MRTP synchronization protocol based on the E2ACE leader election algorithm and our predictive method to compensate for communication delays.

6.2.3. Time of convergence and achievable precision

Figure 12 shows the average maximum pairwise synchronization error of the modules over time for the compared synchronization protocols. During the first hour, the modules were not synchronized and progressively drifted apart. The system reached a synchronization error of more than 40 seconds.

Time of convergence. MRTP, MLE_TPSN and PulseSync centralized protocols converge in a few seconds in the three systems. We recall that MRTP and MLE_TPSN first elect a leader, build a spanning-tree, and then start synchronizing the modules. In PulseSync, modules wait for 5 synchronization periods (i.e., 25 seconds) without hearing a synchronization message before declaring themselves as time masters and trying to synchronize the other nodes. This mechanism causes PulseSync to converge a little bit slower but makes this protocol inherently tolerant to faults.

As expected, ATS, which is an average consensus-based decentralized protocol, converges much more slowly and the time of convergence significantly increases with the system size. In Ball(15), ATS converges only after about 30 minutes of periodic synchronization. WMTS, which is a maximum-value consensus-based protocol, converges more quickly than ATS. But WMTS is still slightly slower than MRTP, MLE_TPSN and PulseSync centralized protocols.

FTSP does not converge in large ensembles of Blinky Blocks. Theoretically, FTSP should have converged in less than 15 minutes in Ball(27) (Maróti et al., 2004). As explained in the related work subsection, FTSP synchronization waves are slowly flooded through the network using asynchronous broadcasts, whereas, in MRTP, MLE_TPSN and PulseSync, the current global time gets quickly disseminated throughout the entire network. This last scheme significantly reduces the impact of clock inaccuracies (due to noise, skew variations, time-increasing errors in the local estimation of the global time) on the synchronization precision and the time of convergence.

Synchronization precision. Figure 13 shows statistics on the maximum pairwise synchronization error after convergence. Unsurprisingly, the synchronization precision of all the protocols decreases with the network size. MRTP, MLE_TPSN and PulseSync which are centralized protocols have a synchronization precision of a few dozens of milliseconds in all the considered systems.

MRTP-E2ACE-PRED is the most precise protocol. As shown in Figure 13, using a central node as the time master improves the average maximum pairwise synchronization error of MRTP by about 0.6 to 3.5 milliseconds in the different ball systems (MRTP-E2ACE-PRED vs MRTP-MIN_ID-PRED). Moreover, the precision improvement increases with the diameter of the ball.

Unsurprisingly, MRTP-MIN_ID-PRED and PulseSync-PRED have in average a similar synchronization precision. It was awaited as the two protocols only differ by the mechanism they used to elect the minimum-identifier node and by their infrastructure (i.e., MRTP uses a breath-first spanning-tree while PulseSync is infrastructure-less and floods the network). However, it must be noted that, in Ball(27), MRTP-MIN_ID-PRED has in average a slightly lower synchronization error. We did not investigate this point but we suspect this could be due to the fact that, in MRTP, a node always gets synchronized by a message that has traveled on the same and shortest path while, in our implementation of PulseSync, synchronization messages can come from different and possibly not shortest paths depending on the network traffic.

As announced in Section 6.1.2, MRTP can effectively synchronize the Ball(27) system, composed of 27,775 modules, to less than 40 milliseconds. Indeed, it synchronizes this system to 17 milliseconds in average and to 24 milliseconds at worst.

6.2.4. Communication efficiency

Number of messages. Figure 14 shows the average number of messages sent per module and its decomposition according to message types. We consider three types of messages: the messages due to the leader election process, the ones due to the tree infrastructure creation and the synchronization messages.

As awaited, ATS and WMTS decentralized synchronization protocols use in average more messages per module than MRTP, MLE_TPSN and PulseSync centralized protocols. In addition, PulseSync, which uses network-wide floodings, generates in average more messages per module than MRTP and MLE_TPSN, which use a tree-like structure. Thus, the message cost induced by both the leader election process and the infrastructure construction is compensated in less than one hour.

Let k denotes the number of messages used by the compensation delay method ($k = 1$ for PRED and $k = 3$ for RTT). In decentralized methods, $2km$ messages are sent per synchronization round, while $k(n-1)$ messages are sent in MRTP and MLE_TPSN, and $2m-(n-1)$ messages are sent in PulseSync-PRED (after the time master election has converged). We recall that $n-1 \leq m$ in connected networks. Because MLE_TPSN uses a round-trip time based method, it generates three times more synchronization messages per synchronization phase than MRTP with PRED. In compact systems, the number of links is more important than the number of nodes. Thus, in these systems, PulseSync generates more messages per synchronization round than MRTP with PRED. However, PulseSync is inherently more tolerant to network failures because synchronization waves are flooded through all links and not only along the links of a spanning tree. Thus, if a link fails but the system remains connected, PulseSync may still be able to synchronize all the modules. Nevertheless, in a spanning-tree, if a link fails, all the nodes of a sub-tree will not get synchronized. MRTP handles this case by re-electing a time master and reconstructing the synchronization tree.

Message queue usage. We measured the maximum message queue size reached by the modules taking into account both the incoming and the outgoing messages. We observed that for any module, the ratio of the maximum reached queue size to the number of neighbors of that module, remains below or equal to three, regardless of the size of the networks for all the protocols except for PulseSync. For PulseSync, the ratio reached the value of 4.5. Thus, nor the leader election process, which involves network-wide flooding(s), nor the actual synchronization phases overwhelm the network. The traffic generated by the synchronization protocols remain well controlled and modules do not require a lot of memory space to store incoming and outgoing messages.

7. Discussion

In this paper, we described the Modular Robot Time Protocol (MRTP), a network-wide time synchronization protocol for modular robots. MRTP is intended to synchronize large-scale and fairly stable systems where changes in the network topology, due for instance to module mobility, or potential module or link failures, are infrequent. Our protocol achieves its performance by combining several mechanisms: distributed central-time-master election, fast and recursive propagation of synchronization waves along the edges of a breadth-first spanning-tree, low-level timestamping and per-hop compensation for communication delays using the most-appropriate method, and clock skew compensation using linear regression.

In MRTP, a dynamically elected central module periodically synchronizes the system. Placing the time master close to the center of the network increases the overall synchronization precision because cumulative errors are made every hop. This strategy is particularly judicious in our context because large-scale modular robots with neighbor-to-neighbor communications tend to exhibit large hop distances. In order to synchronize the system, the time master periodically launches synchronization waves, which are recursively propagated along the edge of a breadth-first spanning-tree. Slave modules propagate these waves to their children in the tree shortly after reception. As explained in (Lenzen et al., 2009), optimal synchronization requires a fast propagation scheme. Also note that using a tree is more communication efficient in compact systems than flooding approaches. Indeed, since there is no broadcast support in the neighbor-to-neighbor communication model, a node has to send an individual copy of a message to all its neighbors in order to broadcast that message. Furthermore, using a breadth-first tree guarantees that synchronization messages always travel on the same and shortest paths. This also leads to better synchronization precision. MRTP performs per-hop synchronization, i.e., a module gets synchronized by an one-hop neighbor. At each hop, the propagated estimation of the current global time is updated to take into account communication delays and time of residence in intermediate modules. Any approach to compensate for these delays can be used in MRTP. Most of the existing approaches use low-level timestamping to suppress the main sources of uncertainty in delay estimations. The best-suited technique to actually use in MRTP depends on the target platform (i.e., the clock precision, its resolution and the communication mechanism) and should be carefully selected, since it has a direct impact on the performance of our protocol, both in terms of precision and communication efficiency. We provided a method to experimentally evaluate the precision of a given approach over multiple hops.

We evaluated our protocol on the Blinky Blocks platform, both on hardware and through simulations. We showed that MRTP can manage systems composed of up to $27,775$ Blinky Blocks. We observed that the synchronization precision depends on the hop distance to the time master, the synchronization periods and the number of synchronization points used for the linear regressions. Furthermore, we showed that MRTP is able to successfully maintain a Blinky Blocks system synchronized to a few milliseconds, using few network resources at runtime, although the Blinky Blocks use low-bitrate communications ($38.4\ kbit/s$) and are equipped with a very low accuracy ($10,000$ parts per million (ppm)) and poor resolution (1 millisecond) clocks. Moreover, we compared MRTP to existing synchronization protocols ported to fit our system model. Simulations results show that MRTP exhibits in average a lower maximum pairwise synchronization error than compared protocols while sending more than half less messages in compact systems.

Our protocol is portable to any modular robot system where modules interact together using only neighbor-to-neighbor communications even if their internal clocks are low precision and have high skew relative to one another. Depending on the time master election procedure, it may also be required that every module has a unique identifier. It must be noted that our protocol can also be used in systems with more precise clocks. It will indeed have two main effects. First, a lower resolution will lead to more precise local clock readings, i.e., more precise message timestamps. Hence, communication delays may be more precisely captured and compensated for, using potentially a different method than the predictive one we use with the Blinky Blocks. Second, a more precise clock implies reduced clock skew, drift (variation of skew) and noise. This can only increase our protocol precision. It must be noted that, even with higher-precision clocks, it is still appropriate to use a linear model to compensate for short-term clock skew. Indeed,

this approach is also commonly used in systems equipped with more precise clocks (e.g., RBS (Elson et al., 2002), FTSP (Maróti et al., 2004), PulseSync (Lenzen et al., 2015), etc.). Consequently, our protocol should also be able to efficiently synchronize systems equipped with higher-precision clocks. We let the evaluation of our protocol in such systems for future works.

8. Future Work

In future work, we plan to test MRTP in large-scale hardware systems running real applications, which have time synchronization requirements and which may potentially generate a significant network and computing load.

In addition, it would be interesting to design more precise methods to compensate for network delays in Blinky Blocks systems. We envision, for instance, to enhance FD with a method that will compensate for the dissemination error after several hops, i.e., when this error has become greater than the resolution of the clock and can effectively be compensated for. Also, different network delay compensation methods can be combined to provide a better estimation of the current global time. In order to not increase the communication load, a same message can carry multiple timestamps inserted by different methods.

Furthermore, MRTP should be tested in other systems that fit our system model. In particular, it will be interesting to evaluate our protocol on hardware systems equipped with more precise clocks.

Moreover, we plan to study time synchronization in highly dynamic modular robotic systems where module mobility and failures may occur frequently. In particular, we will address the problem of time synchronization throughout the process of self-reconfiguration, during which modules move to re-arrange the global shape of the modular robot (e.g., (Piranda and Bourgeois, 2016),(Lakhlef et al., 2014)). MRTP needs to be adapted to efficiently handle such network dynamics, because the frequent re-elections of a central module and the maintenance of the synchronization tree will be too expensive. For now, we suggest to use the high-level framework of the PulseSync protocol (Lenzen et al., 2015) in those systems. This framework is indeed inherently tolerant to module mobility and failures.

9. Acknowledgments

This work has been funded by the "Programmable Matter" ANR project (ANR-16-CE33-0022-02), the French Investissements dAvenir program, the ISITE-BFC project (ANR-15-IDEX-03), the Labex ACTION program (ANR-11-LABX-01-01) and the Mobilitech project.

References

Allan, D.W., 1987. Time and frequency(time-domain) characterization, estimation, and prediction of precision clocks and oscillators. IEEE transactions on ultrasonics, ferroelectrics, and frequency control 34, 647–654.

Amundson, I., Kusy, B., Volgyesi, P., Koutsoukos, X., Ledeczi, A., 2008. Time synchronization in heterogeneous sensor networks, in: Distributed Computing in Sensor Systems. Springer, pp. 17–31.

ATMEL, 2013. XMEGA A3 microcontroller data-sheet.

ATMEL, 2016. AVR1003: using the XMEGA™ clock system.

Baca, J., Ferre, M., Collar, M., Fernandez, J., Aracil, R., 2010. Synchronizing a modular robot colony for cooperative tasks based on intra-inter robot communications, in: Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010, IEEE. pp. 388–393.

Bourgeois, J., Goldstein, S., 2012. Distributed intelligent mems: Progresses and perspectives. ICT Innovations 2011 , 15–25.

Bourgeois, J., Piranda, B., Naz, A., Lakhlef, H., Boillot, N., Mabed, H., Douthaut, D., Tucci, T., 2016. Programmable matter as a cyber-physical conjugation, in: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, IEEE, Budapest, Hungary.

Cheung, T.Y., 1983. Graph traversal techniques and the maximum flow problem in distributed computation. Software Engineering, IEEE Transactions on SE-9, 504–512.

Cristian, F., 1989. Probabilistic clock synchronization. Distributed Computing 3, 146–158.

Dhoutaut, D., Piranda, B., Bourgeois, J., 2013. Efficient simulation of distributed sensing and control environments, in: Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, IEEE. pp. 452–459.

Elson, J., Girod, L., Estrin, D., 2002. Fine-grained network time synchronization using reference broadcasts. ACM SIGOPS Operating Systems Review 36, 147–163.

Etzlinger, B., Wymeersch, H., Springer, A., 2014. Cooperative synchronization in wireless networks. IEEE Transactions on Signal Processing 62, 2837–2849.

Ferrari, F., Zimmerling, M., Thiele, L., Saukh, O., 2011. Efficient network flooding and time synchronization with glossy, in: Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on, IEEE. pp. 73–84.

Flajolet, P., Fusy, É., Gandouet, O., Meunier, F., 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm, in: AofA: Analysis of Algorithms, Discrete Mathematics and Theoretical Computer Science. pp. 137–156.

Flury, R., Wattenhofer, R., 2010. Slotted programming for sensor networks, in: Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks, ACM. pp. 24–34.

Ganeriwal, S., Kumar, R., Srivastava, M.B., 2003. Timing-sync protocol for sensor networks, in: Proceedings of the 1st international conference on Embedded networked sensor systems, ACM. pp. 138–149.

Goldstein, S.C., Mowry, T.C., 2004. Claytronics: An instance of programmable matter, in: Wild and Crazy Ideas Session of ASPLOS, Boston, MA.

Gusella, R., Zatt, S., 1989. The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3 bsd. Software Engineering, IEEE Transactions on 15, 847–853.

Habibi, A., Dedu, E., Bourgeois, J., Laurent, G.J., Le Fort-Piat, N., 2012. Distributed pneumatic mems for fast conveyance of fragile objects, in: JRWRTC'12, 6th Junior Researcher Workshop on Real-Time Computing, joint to RNTS'12, the Int. Conf. on Real-Time and Network Systems, pp. 33–36.

He, J., Cheng, P., Shi, L., Chen, J., Sun, Y., 2014a. Time synchronization in wsns: A maximum-value-based consensus approach. IEEE Transactions on Automatic Control 59, 660–675.

He, J., Li, H., Chen, J., Cheng, P., 2014b. Study of consensus-based time synchronization in wireless sensor networks. ISA transactions 53, 347–357.

Hosseinmardi, H., Correll, N., Han, R., 2012. Bloom filter-based ad hoc multicast communication in cyber-physical systems and computational materials. Wireless algorithms, systems, and applications , 595–606.

IEEE, 2008. IEEE 1588-2008: Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. Technical Report. IEEE.

Kim, C., Wu, M., 2013. Leader election on tree-based centrality in ad hoc networks. Telecommunication Systems 52, 661–670.

Kim, H., Ma, X., Hamilton, B.R., 2012. Tracking low-precision clocks with time-varying drifts using kalman filtering. IEEE/ACM Transactions on Networking (TON) 20, 257–270.

Kirby, B.T., Ashley-Rollman, M., Goldstein, S.C., 2011. Blinky blocks: a physical ensemble programming platform, in: CHI '11 Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, USA. pp. 1111–1116.

Kokaji, S., Murata, S., Kurokawa, H., Tomita, K., 1996. Clock synchronization mechanisms for a distributed autonomous system. J. Robotics and Mechatronics 8, 427–434.

Kusy, B., 2007. Spatiotemporal coordination in wireless sensor networks. Ph.D. thesis. Vanderbilt University. Nashville, TN, USA.

Lakhlef, H., Mabed, H., Bourgeois, J., 2014. Optimization of the logical topology for mobile mems networks. Journal of Network and Computer Applications 42, 163–177.

Leng, M., Wu, Y.C., 2010. On clock synchronization algorithms for wireless sensor networks under unknown delay. IEEE Transactions on Vehicular Technology 59, 182–190.

Lenzen, C., Sommer, P., Wattenhofer, R., 2009. Optimal clock synchronization in networks, in: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, ACM. pp. 225–238.

Lenzen, C., Sommer, P., Wattenhofer, R., 2015. Pulsesync: An efficient and scalable clock synchronization protocol. IEEE/ACM Transactions on Networking (TON) 23, 717–727.

Li, Q., Rus, D., 2006. Global clock synchronization in sensor networks. Computers, IEEE Transactions on 55, 214–226.

Malpani, N., Welch, J.L., Vaidya, N., 2000. Leader election algorithms for mobile ad hoc networks, in: Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications, ACM. pp. 96–103.

Maróti, M., Kusy, B., Simon, G., Lédeczi, Á., 2004. The flooding time synchronization protocol, in: Proceedings of the 2nd international conference on Embedded networked sensor systems, ACM. pp. 39–49.

McEvoy, M.A., Correll, N., 2015. Materials that couple sensing, actuation, computation, and communication. Science 347, 1261689.

Mills, D.L., 1991. Internet time synchronization: the network time protocol. Communications, IEEE Transactions on 39, 1482–1493.

Naz, A., Piranda, B., Bourgeois, J., Goldstein, S.C., 2015a. Blinky Blocks Time Protocol (BBTP) : protocole de synchronisation des horloges internes de dispositifs embarqués. Research Report RR-FEMTO-ST-2671. FEMTO-ST.

Naz, A., Piranda, B., Goldstein, S.C., Bourgeois, J., 2015b. ABC-Center: Approximate-center election in modular robots, in: IROS 2015, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Hamburg, Germany. pp. 2951–2957.

Naz, A., Piranda, B., Goldstein, S.C., Bourgeois, J., 2016a. Approximate-centroid election in large-scale distributed embedded systems, in: AINA 2016, 30th IEEE Int. Conf. on Advanced Information Networking and Applications, IEEE, Crans-Montana, Switzerland. pp. 548–556.

Naz, A., Piranda, B., Goldstein, S.C., Bourgeois, J., 2016b. A time synchronization protocol for modular robots, in: PDP 2016, 24th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing, IEEE, Heraklion Crete, Greece. pp. 109–118.

Naz, A., Piranda, B., Tucci, T., Goldstein, S.C., Bourgeois, J., 2016c. Network characterization of lattice-based modular robots with neighbor-to-neighbor, in: 2016 13th International Symposium on Distributed Autonomous Robotic Systems (DARS), Springer, London, UK.

Noh, K.L., Chaudhari, Q.M., Serpedin, E., Suter, B.W., 2007. Novel clock phase offset and skew estimation using two-way timing message exchanges for wireless sensor networks. IEEE transactions on communications 55, 766–777.

Piranda, B., Bourgeois, J., 2016. A distributed algorithm for reconfiguration of lattice-based modular self-reconfigurable robots, in: PDP 2016, 24th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing, IEEE, Heraklion Crete, Greece. pp. 1–9.

Raynal, M., 2013. Distributed algorithms for message-passing systems. volume 500. Springer.

Römer, K., Blum, P., Meier, L., 2005. Time synchronization and calibration in wireless sensor networks. Handbook of sensor networks: Algorithms and architectures 49, 199.

Şahin, E., 2004. Swarm robotics: From sources of inspiration to domains of application, in: International workshop on swarm robotics, Springer. pp. 10–20.

Schenato, L., Fiorentin, F., 2011. Average timesynch: A consensus-based protocol for clock synchronization in wireless sensor networks. Automatica 47, 1878–1886.

Sommer, P., Wattenhofer, R., 2009. Gradient clock syn-

chronization in wireless sensor networks, in: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks, IEEE Computer Society. pp. 37–48.

Su, W., Akyildiz, I.F., 2005. Time-diffusion synchronization protocol for wireless sensor networks. Networking, IEEE/ACM Transactions on 13, 384–397.

Vasudevan, S., Kurose, J., Towsley, D., 2004. Design and analysis of a leader election algorithm for mobile ad hoc networks, in: Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on, IEEE. pp. 350–360.

Yim, M., White, P., Park, M., Sastra, J., 2009. Modular self-reconfigurable robots, in: Encyclopedia of complexity and systems science. Springer, pp. 5618–5631.

Name	Domain	Architecture	Infrastructure	Synchronization Technique	Clock Skew Compensation
NTP (Mills, 1991)	Computer Networks	Master/Slave Master(s): pre-configured	Tree	(Multi-hop) round-trip messages with frame-level timestamps and statistics	Phase-locked and/or frequency-locked loops
PTP (IEEE, 2008)	Computer Networks	Master/slave Master: clock quality based election	Tree	Round-trip messages with low-level (data-link to physical layer) timestamps and per-hop delay compensation	
TPSN (Ganeriwal et al., 2003)	Sensor Networks	Master/slave	Tree	Recursive per-hop synchronization. Round-trip messages with frame-level timestamps	/
TPSN (Ganeriwal et al., 2003) + MLE (Leng and Wu, 2010)	Sensor Networks	Master/slave	Tree	Recursive per-hop synchronization. Round-trip messages with frame-level timestamps and statistics	Linear model with maximum likelihood estimators
TDP (Su and Akyildiz, 2005)	Sensor Networks	Masters/Slave multiple changing masters: clock quality based election	/	Recursive per-hop synchronization. Bidirectional round-trip messages with statistics	/
RBS (Elson et al., 2002)	Sensor Networks	Master/Slave	Broadcast-domain based clustering	Reference broadcast	Linear model with least-square linear regression
FTSP (Maróti et al., 2004)	Sensor Networks	Master/slave Master: id-based implicit election	/	Periodic asynchronous broadcasts. Unidirectional broadcast with byte-level timestamps and statistics	Linear model with least-square linear regression
RATS (Kusy, 2007)	Sensor Networks	Master/Slave Master: pre-configured	/	Recursive per-hop synchronization. Unidirectional broadcast with byte-level timestamps and statistics	Linear model with least-square regression
Pulse-Sync (Lenzen et al., 2009, 2015)	Sensor Networks	Master/slave Master: id-based implicit election	/	Recursive per-hop synchronization. Unidirectional broadcast with byte-level timestamps and statistics	Linear model with least-square linear regression
AD (Li and Rus, 2006)	Sensor Networks	Fully distributed	/	Average-based consensus	/
GTSP (Sommer and Wattenhofer, 2009)	Sensor Networks	Fully distributed	/	Average-based consensus. Unidirectional broadcast with byte-level timestamps and statistics	Linear model with an averaging technique
ATS (Schenato and Fiorentin, 2011)	Sensor Networks	Fully distributed	/	Average-based consensus. Unidirectional broadcast with byte-level timestamps	Linear model with an averaging technique
MTS and its variants (He et al., 2014a,b)	Sensor Networks	Fully distributed	/	Extremum-value based consensus. Unidirectional broadcast with byte-level timestamps	Linear model with possibly an averaging technique
BP and MF (Etzlinger et al., 2014)	Sensor Networks	Master/Slave or fully distributed	/	Belief propagation and mean field. Single-hop bidirectional messages with frame-level timestamps	Linear model with maximum a posteriori estimators
Our Contribution: MRTP	Modular Robotic	Master/Slave Master: centrality-based election	Tree	Recursive per-hop synchronization. Selection of the best-suited communication delay compensation method for the target system	Linear model with least-square linear regression

Table 1: Related work summary.

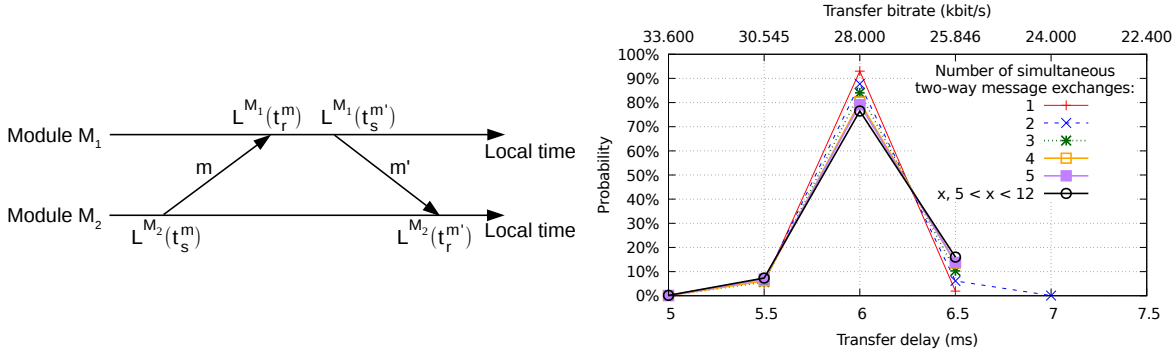

Figure 5: On the left: scheme of a two-way message exchange between two blocks. On the right: transfer delay/rate distribution of 21-byte long frames.

Parameters			Value
Timeouts	Triggering time (s)		$\mathcal{U}(0, 500 \cdot 10^{-6})$
	Processing time (s)		$\mathcal{U}(250 \cdot 10^{-6}, 300 \cdot 10^{-6})$
Messages	Queue occupancy at arrival	Light load	neighborhood management
		Moderate load	neighborhood management + $\mathcal{P}(1)$
	Transfer rate ($kbit/s$)	Sparse systems (e.g., line system)	$\mathcal{N}(28.134, 0.660)$
		Intermediate systems (e.g., L-shaped systems)	$\mathcal{N}(28.085, 0.938)$
		Compact systems (e.g., ball systems)	$\mathcal{N}(27.696, 1.143)$
	Processing time (s)	Low complexity	$\mathcal{U}(250 \cdot 10^{-6}, 300 \cdot 10^{-6})$
		Medium complexity	$\mathcal{U}(475 \cdot 10^{-6}, 525 \cdot 10^{-6})$

Table 3: Communication model. $\mathcal{N}(\mu, \sigma)$ refers to the normal probabilistic law with μ mean and σ standard-deviation. $\mathcal{U}(l, u)$ refers to the uniform probabilistic law with minimum value l and maximum value u. $\mathcal{P}(\lambda)$ refers to the Poisson probabilistic law with λ mean.

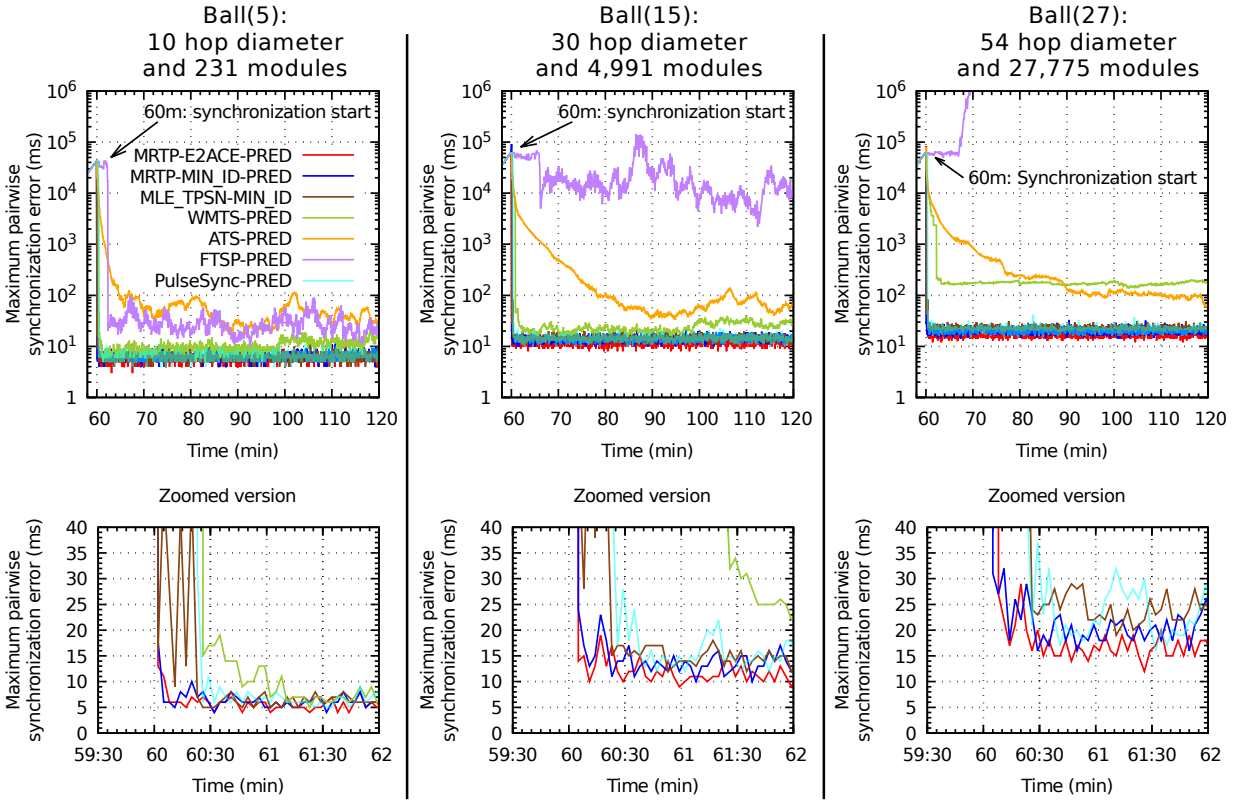

Figure 12: Maximum pairwise synchronization error over time. This figure shows both the time of convergence and the achievable precision for each protocol on the different Ball systems.

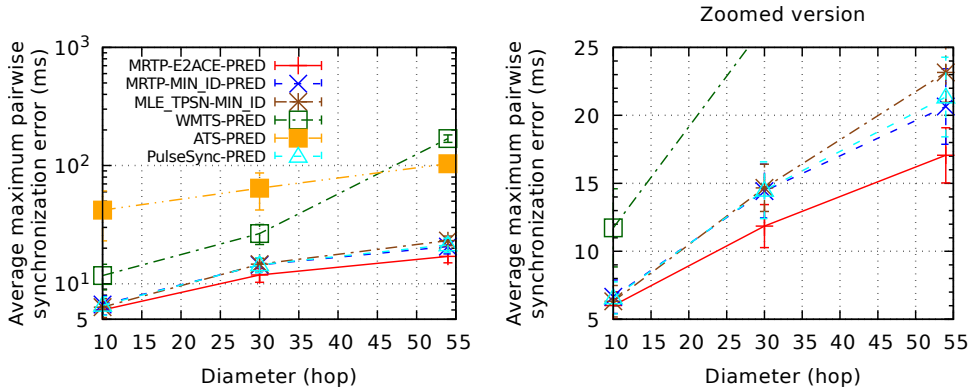

Figure 13: Average maximum pairwise synchronization error on the last 30 minutes of experiment (± standard-deviation).

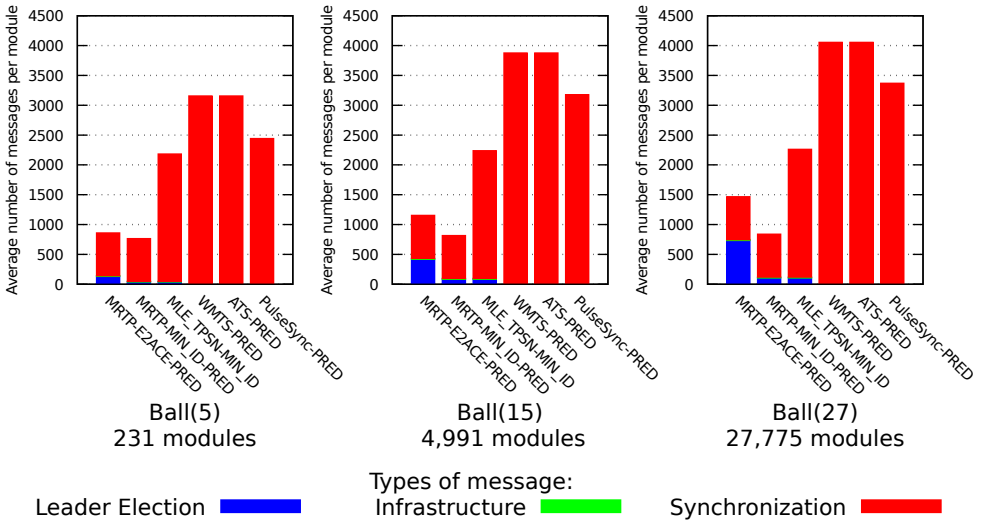

Figure 14: Average number of messages sent per module.