# Time in Cyber-Physical Systems

Aviral Shrivastava*, Patricia Derler†, Ya-Shian Li Baboud‡, Kevin Stanton§
Mohammad Khayatian*, Hugo A. Andrade†, Marc Weiss‡, John Eidson¶, Sundeep Chandhoke†
*Arizona State University, †National Instruments, §Intel Corporation
‡National Institute of Standards and Technology, ¶University of California, Berkeley

***Abstract*—Many modern cyber-physical systems (CPS), especially industrial automation systems, require the actions of multiple computational systems to be performed at much higher rates and more tightly synchronized than is possible with ad hoc designs. Time is the common entity that computing and physical systems in CPS share, and correct interfacing of that is essential to flawless functionality of a CPS. Fundamental research is needed on ways to synchronize clocks of computing systems to a high degree, and on design methods that enable building blocks of CPS to perform actions at specified times. To realize the potential of CPS in the coming decades, suitable ways to specify distributed CPS applications are needed, including their timing requirements, ways to specify the timing of the CPS components (e.g. sensors, actuators, computing platform), timing analysis to determine if the application design is possible using the components, confident top-down design methodologies that can ensure that the system meets its timing requirements, and ways and methodologies to test and verify that the system meets the timing requirements. Furthermore, strategies for securing timing need to be carefully considered at every CPS design stage and not simply added on. This paper exposes these challenges of CPS development, points out limitations of previous approaches, and provides some research directions towards solving these challenges.**

## I. Introduction

Next-generation cyber-physical systems (CPS) need to provide seamless coordination of the cyber (software and hardware) components for enabling autonomous, self-organizing applications that are dynamically responsive to system demands. The Internet of Things (IoT) surrounds us with sensors, computing, communication and control components to provide capabilities of translating real-time measurements into actionable intelligence for unparalleled awareness, coordination, interaction and efficiency. The notion of efficiency is often equated with time and timeliness. Timely detection of anomalies in a system, maximizing the use of energy at times where tariffs are lowest, or locating a victim in a disaster are just a few examples of the need for time awareness in CPS for coordinating potentially ad hoc sources of information and providing timely actionable intelligence to optimize the outcome.


CODES/ISSS '16, October 01-07, 2016, Pittsburgh, PA, USA
 ISBN 978-1-4503-4483-8/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2968456.2974012

One example of a complex, spatially expansive, and decentralized CPS is the power system. Achieving the green imperative entails efficient management of intermittent energy sources (wind and solar), storage (batteries), and loads (electric vehicles). The reliance on variable generation sources requires a responsive, flexible, adaptive, local, and fast management model [1]. Enabling the Smart Grid, as in other complex heterogeneous CPS systems, requires systems that are time synchronized and time aware. The accuracy of the timing and the timeliness of the data directly impact the accuracy of the measurement results [2], [3], [4].

A proper design methodology is needed in order to design robust, coordinated and time-sensitive CPS that meet the reliability and resiliency demands, and remain stable to transient disturbances and recover from faults in real-time [5]. Figure 1 shows the parts of the problem, and outlines the steps to developing such a methodology.

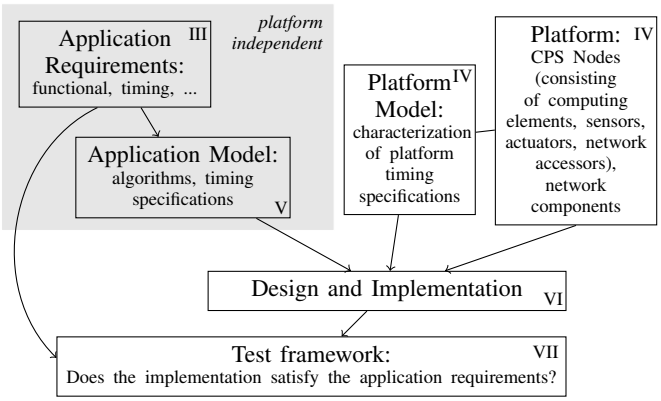



Figure 1: CPS Design Process: from capturing application requirements to testing

The first step towards confident design is to specify the timing requirements of the CPS application, and the timing specifications of the CPS components. Adequate consideration also needs to be given at this stage to the extent to which timing services need to be secured. These requirements and specifications can derive the rest of the system design, analysis and testing. Timing requirements of CPS (section III) vary widely depending on application, from requiring no synchronization between the CPS nodes to the CPS nodes performing the same action at the same moment to within microsecond or even sub-microsecond level accuracy. Achieving the timing

requirements in complex, ad hoc systems requires explicit timing specifications of the CPS components (section IV), i.e., the components that the CPS is built of, e.g., sensors, actuators, computing platform, network. While the timing specifications of sensors and actuators are more clearly defined and easier to specify, the timing specifications of the computing platforms are much more complex and difficult to analyze and specify. Given satisfiability of the application's timing requirements, we need a time-aware design methodology (section VI) that can map the CPS application onto the platform in such a way that all the timing requirements of the application will be met. This may require synchronization among the subsets of CPS nodes, and making sure that each CPS node is able to perform its tasks in a timely manner. However, even after a confident design, there is a need to design tests that can determine if all the timing requirements are being met (section VII). Since exhaustive testing may be infeasible, it will be important to design systems which provide some monotonic properties with respect to time, so that the test space can be reduced.

## II. The timing challenge in CPS

One of the key challenges in designing distributed CPS is establishing a common notion of time between the physical world where time is continuous and the computational system, where time is incremented in discrete units. In order to achieve tight orchestration of system components to meet time-sensitive application demands, the CPS needs a common physical time scale, most commonly achieved by clock synchronization, and a means to specify temporal behavior.

Clock synchronization can improve distributed algorithms by replacing communication with local computation [6]. Clock synchronization is critical for data fusion as embedded systems provide distributed measurements to monitor and react to physical behavior. The challenge in clock synchronization over a large spatial expanse lies first in the ability to account for the delay in transmitted clock signals, and second in the ability for components to reliably receive timing data and be resilient to loss of a timing source.

Beyond sensing, a CPS involves computing and actuating in control loops. Computing and communication processes must be achieved within time limits and control commands are typically time-sensitive and require time-aware networks for transmitting them with bounded latencies. With current computers and networks that are not time aware, it is difficult to achieve bounded latencies for complex distributed CPS.

Another critical aspect is the ability to specify the temporal behavior of the CPS. The intelligence behind the CPS lies in the ability to make timely updates of system models to enable sufficient dynamic understanding in order to predict behavior and take action. In power systems, for example, the amount of power generated from a renewable source or the amount of demand on loads can vary on a relatively fast time scale [2]. Verifiable semantics are needed to describe temporal ordering, frequency, latency, and simultaneity in order to coordinate all the components of the system and achieve the application goals.

Timing constraints are often implicit in the design of embedded systems [7]. Existing methods include costly customized solutions where correct timing behavior is achieved, through trial and error and only on a specific computing and communication platform. The disconnect between timing requirements and system specifications makes it difficult to achieve consistency and correctness in the temporal behavior of a CPS across heterogeneous platforms. For example, a CPS can be comprised of a network of *CPS nodes* or embedded systems, each interacting with the physical plant to measure, analyze, and control it over a network. Heterogeneous CPS nodes include sensing, computing, communicating, and actuating platforms capable of self-discovery and self-organization to determine how to most effectively monitor, detect, respond, repair, and recover. Explicit design of applications that can satisfy temporal requirements can only be feasible if each CPS node in the system, including the communication network, are time aware.

Temporal logic languages provide the syntax and semantics to enable formal manipulations to prove a timing requirement is satisfiable given a particular system model. However, in practice, formal logic can be a challenge to understand and difficult to apply. Achieving tight timing requirements typically require custom built designs specific to a single platform. The challenge is to be able to specify the temporal behavior in a high-level language and be able to readily port to other distributed embedded platforms, where tests can be automatically generated from the specifications and the compiler can synthesize the software specifications, operating system demands and the hardware capabilities to determine the schedulability of the application requirements [8].

Research is ongoing to develop semantics and frameworks for high-level abstraction of temporal behavior between application and platform models [9]. One of the key challenges is reducing the gap between the simplifying assumptions in the abstractions and the actual timing properties of the platform [10]. CPS design is a multidisciplinary effort and timing typically affects more than just one discipline, where inconsistencies can arise between application developers and platform architects/engineers. Design contracts provide a means to specify and negotiate timing constraints between components (in literature also referred to as horizontal contracts [11]) as well as across design disciplines (see vertical contracts [11] and design contracts for timing in CPS [7]). Without a high-level abstraction of timing constraints, it is difficult to test if the requirements on temporal behavior can be achieved. The primary challenges in the ability to design and verify correct temporal behavior in CPS are: (1) to realize the explicit specification of temporal constraints in a CPS by the designers, (2) to provide feedback during the design and implementation phases, and (3) to reliably ensure temporal correctness of application behavior on distributed platforms.

## III. Timing Requirements of CPS Applications

Timing requirements in CPS can be categorized as follows:

**Frequency constraints.** Physical phenomena (e.g. weather, electrical arcing, particle behavior, physiological response) and human behavior can be difficult to predict and can vary over time. Sensors need to be able to sample at a sufficient rate. Based on the Nyquist Theorem, the sampling rate must be at least twice the maximum frequency, or twice the highest analog frequency component to be able to capture rapid, transient events. *Frequency constraints* are necessary to ensure sampling requirements are met. In addition, an embedded system typically has sampling frequency requirements which allow the capture of rapid transients or the assessment of the progression and stability of the system. Application *frequency constraints* are typically implemented with interrupt-based task scheduling. Other frequency constraints are related to coordinating behavior, such as multiplexing signals from remote sources. This is generally solved with frequency locked-loops.

**Chronological constraints.** Information from heterogeneous sources in distributed systems where precision and time scales vary must be merged unambiguously such that all observers agree on the sequence of events. Semantics enabling a consistent *chronological constraint* are necessary for establishing this sequence of events. By providing the semantics to establish a chronological order, information about a component or event can be characterized as a past, present or future, and all observers of the component can agree on the chronological relationship between events. Typical chronological constraints require a monotonically increasing and continuous, common time scale with sufficient accuracy. Providing semantic ordering also determines *causality*.

**Simultaneity constraints.** These constraints are needed to ensure that two events occur at the same time to all observers in the system [12]. Enabling measurement and control systems to be triggered at the same time allows coordination, detection and localization of an event of interest. The understanding of natural synchrony can facilitate the ability to understand, model, and replicate the parameters and processes of achieving synchronous behavior of an ad hoc group of components in the engineered CPS [13]. CPS components may be required to act in synchrony. In computer science, concurrency requires processes to run simultaneously. In power systems, fault location applications require measurements that occur as closely to each other as possible [14]. In particle physics experiments, clock synchronization is required to ensure all detectors can be triggered simultaneously to collect data at specific instants in time [15]. The higher the precision, the lower the uncertainty of distributed measurement.

**Latency constraints.** In time-sensitive applications, sensor information and knowledge computed from the sensors is valid for only a specific temporal interval before it must be acted upon, resulting in bounded or fixed latency constraints on communication and computation. Timeliness, or the temporal limits of the application to communicate information or execute an action can be described through *latency constraints*. Programmers define a function sequentially from its input to its output. A function's output at time $t$ relies on input from before time $t$. Temporal dependency requirements on the input arriving before time $t$ via the communication network is needed to ensure the measurement point would be temporally valid for the function. Ensuring the predictability of an application's execution time also requires bounded latencies on the response time of the program and the reaction time of its environment [16]. Latency can be viewed as both a bounded requirement, such as *deadlines*, or a means to derive critical information, such as time and location. Traveling wave fault detection and fault location requires precise synchronization to locate a fault within a certain distance.

**Temporal assurance constraints.** General trust disciplines relating to CPS include security, resilience, safety, reliability, and privacy, with relative importance dependent on application domain. Secure timing is vital to the proper functioning of the CPS and must be specified at the requirements stage for the system. Security of a timing signal requires security of both the physical timing signal and the data associated with the signal. Security of the data in a timing signal needs to be verified for source (*authentication traceability*) as well as *integrity*. Similarly, the application dependent upon synchronized time signal needs to know both that the physical signal came from the correct source, and that the transmission delay has not been tampered with. The dependence on Global Navigation Satellite Systems (GNSS) for timing services means that designers need to be aware of its vulnerabilities which can compromise *availability* and *integrity*. Strategies to counteract jamming-both unintentional and intentional, and spoofing need to be deployed.

Constraints on the assurance strategies include the time it takes to detect the timing anomaly (*detection latency*), the time the system can synchronize to a sufficient precision without a traceable time source (*holdover*), and the time it takes for the system to converge once a traceable source is available (*recovery time*).

## IV. Timing Specifications of CPS Platform

The CPS platform is comprised of *hardware components* (sensors, actuators, computing and communication systems) and *systems software* (operating systems and device drivers), where the platform timing specifications can be characterized to enable the compiler tools to determine whether the application model's timing requirements can be satisfied.

Timing specifications of the CPS node's clock are critical to understanding the accuracy of the measurements and other outputs derived from the clock. Errors in the clock signal or significant performance degradation can rapidly induce faulty measurements. Clock fault detection methodologies such as a Dynamic Allan Variance (DAVAR) [17] rely on an accurate characterization of the clock.

A clock can be characterized by its absolute time (local clock accuracy to a traceable reference), relative time (local clock offset from peers), drift, stability (often characterized by the Allan Variance (AVAR) or Allan Deviation (ADEV), which computes the frequency instability given a specified period), and maximum time interval error (MTIE), with respect to the system reference and resolution of the local clock.

Hardware and systems software, including the operating system and device drivers must have a means to communicate the input and output time intervals. Depending on the mechanism for enforcing input to output intervals, temporal specifications can be categorized as the maximum latency if using a time-triggered mechanism such as Giotto [18] or an event-triggered one like PTIDES [19]. In addition, the accuracy of the latency measurement should be considered if the CPS node itself determines the timing.

In analog control, the combination of the cyber timing and IO latencies determine the loop phase characteristics and delay as well as response times, performance, stability, etc.

A CPS comprises input and output devices as well as computation platforms and communication networks which all have specific timing constraints. A common time specifications for digital inputs is a bounded time interval between two consecutive events that reflects recovery time of transducer and detection circuit. For periodic digital inputs, temporal specifications includes period, period accuracy and phase which are related to the nature of events generator or time-triggered sampling rate of application. It is also important to note the uncertainty variables, including sampling jitter. For sporadic signals, statistics attributed to the number of events in a given time interval including mean, maximum, and minimum should be considered. This reflects the capacity of the input mechanism to handle a bursty sequence of events. Transient delay or the time interval between actual occurrence of the event and when the information signaling the event is available to the cyber portion of a CPS reflects latency specification. For analog inputs, temporal constraints include bandwidth of transducer, resolution of analog to digital converter (ADC), conversion time and sampling period of ADC.

For digital output devices, temporal specifications include minimum time between two consecutive actions which reflects recovery time of actuator as well as signal generation circuit. Similar to input devices, for periodic actions, period, period accuracy, and phase of actuation should be considered which are related to the nature of an events generator or the time-triggered sampling rate of a hold circuit. If action is sporadic, the maximum number of events in a given time interval matters. Burst capability reflects the capacity of the output mechanism to handle a burst sequence of events. Another useful specification is Output latency or the time interval between actual occurrence of the event and when the information signaling the event is executed by the cyber portion of a CPS. For analog output devices, time constraints include frequency response, Digital to Analog Converter (DAC) resolution, DAC conversion time, DAC hold type and rate. For both input and output devices, time sensitive components require specification of precision and accuracy of occurrence time.

The timing of a computing platform is most complex, and depends on both, hardware and software aspects. Hardware specifications include CPU instructions per cycle, worst-case memory latency and bandwidth, bus latency and bandwidth, memory size, cache size, cache associativity and replacement policies. Software specifications that affect system timing are composed of operating system configuration, multi-thread scheduling mechanism, file access method, driver configuration, and virtualization. Parameters like minimum inter-arrival time and worst-case execution time of interrupts can affect the interrupt latency of the platform. The minimum required time between input to output computations can point to the time required for other application threads, garbage collection, initialization, etc. For periodic computations, bounded period, period accuracy and computation cycle can reflect the computation engine capacity. Similarly, for sporadic computations, maximum number of computations in a given interval can be categorized as timing specification. Specification of sporadic computation reflects the capacity of the computation mechanism to handle a burst of computations. Finally, computation latency specifies the time interval between when the input information is available to the cyber portion of a CPS and when the needed information is delivered to the output mechanism.

Network delays are an aggregation of propagation, serialization, queuing and processing delays and are influenced by the bandwidth, traffic patterns, routing policies, and time awareness of the nodes. Network temporal constraints are based on the delays tolerated including round-trip delay, worst case delay *time out*, and packet delay variation (PDV). The round-trip delay specifies the communication delay to and from two CPS nodes. Some applications require a maximum time on delay, or a message will time out. Time outs can lead to a data being lost or invalid. Lastly, the non-determinism of the delay or the tolerance of the CPS application to PDV needs to be specified. PDV statistics calculated over a specified *period* can include minimum, maximum, mean and ratio of maximum to minimum.

## V. Timing Specification in Languages

Typically, the first step in the design of the cyber part of a CPS is the definition of the functional model of the application which entails algorithm development. In many traditional development processes, the next step is to implement these algorithms on hardware and then test and tweak them until the timing is "just right". This leads to brittle designs and unintended emergent behavior when composing different components or systems. We argue that instead, the next step must be the development of a model that contains functional as well as timing specifications. Such a model contains the same algorithms, and in addition, timing specifications in places where it matters; i.e. places which are observable from the outside. These places are the inputs from the environment (sensors) and the outputs to the environment (actuators). The time when inputs are read and when outputs are written is crucial to the overall behavior of the CPS, since changes in the timing can lead to changes in the overall system behavior.

At this point the model is still platform independent, and we assume execution is instantaneous; i.e. execution time is zero. As a result, the behavior of the model is deterministic with respect to value and time. This allows to specify the timing behavior using separate nodes, and that makes the timing analysis independent of the functionality and easier.

A correct implementation of this model must exhibit the same value and timing behavior as the platform independent model. This is further explained in section VI on correct-by-construction methodologies. In order to evaluate whether the model can be implemented on a given platform, an in-depth characterization of the timing behavior of the platform must be performed. Properties such as execution time, communication time, scheduling overhead and network latency must be computed and bounds as well as jitter have to be determined. The model is then enhanced with platform dependent constraints.

Timing specifications must be intuitive and natural, without cluttering the model with unnecessary complexity. Timing specification can be part of the language such as Giotto [18] or PTIDES [19], annotations on existing languages, or separate from the model, for example by using logic such as STL (signal temporal logic) [20].

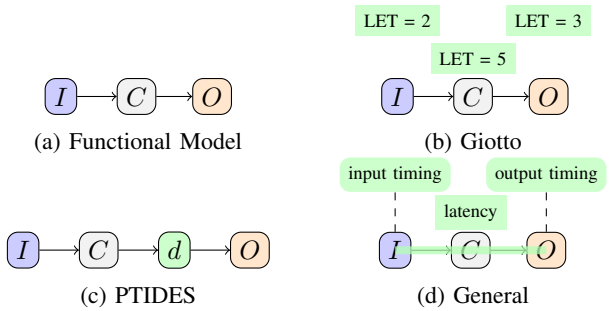
In Giotto, a system is described as a set of tasks where every task has a logical execution time (LET). The start and end of the LET are mapped to real, physical time. Inputs are read in the beginning, outputs are written in the end. When the task executes does not matter as long as it is some time during the LET. As a result, input-output (I/O) is done at well-defined, statically determined times related to task executions and the latency between inputs and outputs is described by the logical execution times on the path. In a Giotto system, not only the inputs and outputs to and from the physical system are defined but also the communication between tasks.

The PTIDES Model of Computation (MoC) extends the discrete event MoC with a notion of physical time in addition to the logical time which is used to order events. Physical and logical time is carefully related only where necessary: at the IO. Since all inputs are event-triggered, the timing of those is not configured in a PTIDES model. The latency between inputs and outputs is well defined via explicit delays on causal paths, thus the timing of outputs with respect to inputs is well defined.

The STL logic deals with properties related to the order of discrete events and the temporal distance between them, where "events" correspond to changes in the satisfaction of some predicate (e.g., threshold crossing) over the real variables. Reading inputs and writing outputs are events that can be related in logic formulas.

Figure 2a shows a simple functional model comprising of three nodes: the input $I$ that receives inputs from the environment and does some simple processing such as filtering, and the output $O$ that actuates after some processing, and a computation $C$ that consumes inputs from $I$ and produces values for $O$. Figure 2b exemplifies Giotto timing specifications where every task is associated with a logical execution time (LET). PTIDES timing specifications are shown in Figure 2c, where a delay $d$ is added between $I$ and $O$ to describe the time delay on this path. Whether this delay is added between $I$ and $C$ or $C$ and $O$ or split up and added in multiple places does not matter but can lead to some confusion for the developer.

A generalization of these timing specifications for an arbitrary MoC is shown in Figure 2d. Here, the timing of the input side effect, the exact time of the interaction with the environment, as well as the timing of the output side effect is configured. These timing configurations can be static, for example, periodic, or dynamic, if IO is read or written upon occurrence of observable events. The latency specification is performed on a causal path. Note that it is not always necessary and may even be redundant to specify the timing of inputs, outputs and the latency on the path. These timing specifications interfere with the MoC in that they add constraints on when certain nodes (most importantly IO nodes) can or have to execute.



(a) Functional Model (b) Giotto (c) PTIDES (d) General

Figure 2: Timing specifications are absent in pure functional models (a) and are implemented as describe execution time bounds on every task in Giotto (b) or bounds on causal paths in PTIDES (c). In general timing specifications are necessary for IO timing and path latencies (d)

## VI. Time Aware CPS Design and Implementation

In general, the computing system in a CPS is distributed, with each computing site referred to as a CPS node. A CPS node performs computations and interacts with a part of the plant via sensors and actuators and communicates with other CPS nodes via a network. In order to derive meaning out of the sensing information, and to correctly control a plant – as outlined in section III – a variety of timing and synchronization requirements must be met:

### A. Accurate synchronization among CPS nodes

The first requirement of implementing a time aware design is acquiring clock synchronization over a CPS network. The synchronization requirements for the CPS may vary dramatically, depending on the application. The synchronization requirements of CPS nodes may vary in time, space, and the level of synchronization. Space refers to the fact that only a subset of the nodes may need to be synchronized, time refers to the fact that this subset may change with time, and level of synchronization refers to whether we want second -accurate synchronizations, or millisecond level, or microsecond level. Furthermore, the synchronization requirement may be local (among the CPS nodes), or against a pre-defined (say UTC) reference. Some of the common synchronization methods that are used in distributed CPS are NTP, PTP and GNSS.

*NTP* or Network Time Protocol (NTP) is one of the oldest synchronization protocols that is used widely over the

Internet [21]. NTP uses a 64 bit time-stamp encapsulation for packets and it cancels transmission delays using a two-way synchronization protocol with clients requesting time from a server. While NTP is useful for wired/wireless CPS nodes networked through the public internet, it has low synchronization accuracy due to asymmetries and PDV between the forward and reverse network delays and unknown or random delays between packet departure time stamps and actual times of departure. It also looses accuracy along the chain of synchronization. So, if CPS node B synchronizes with CPS node A, and then CPS node C synchronizes with CPS node B, then the CPS node C will accrue the inaccuracy of B's synchronization. Therefore, when using NTP, typically all the CPS nodes must synchronize to a node that serves as a dedicated timer server for everyone else.

*GNSS* uses a time reference for its navigation solution. Since a GNSS receiver uses time-based triangulation of RF signals, it's able to maintain highly accurate time and hence an accurate synchronization reference among CPS nodes. The best part is that all the CPS nodes can achieve independent synchronization to UTC, however, GNSS signals are extremely weak, and therefore have limited coverage for indoor applications and are subject to both intentional and unintentional interference, especially in the urban environment.

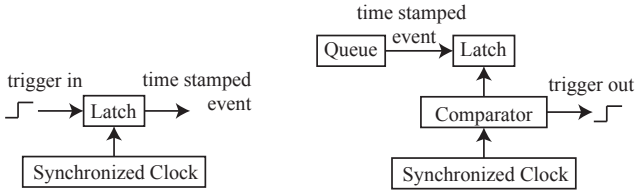
*PTP:* PTP operates using bi-directional time transfer based on a master/slave architecture. PTP can be more accurate than NTP because it provides standard mechanisms for eliminating various sources of time errors, however support must be in place both in the end nodes and in the network elements [22], [23]. Although PTP can deliver highly accurate timing, asymmetric forward and reverse path delays over legacy networks is a major concern, as all synchronization protocols rely on round-trip delay time to compute one-way delay.

CPS that require sub-nanosecond accuracy can employ, White Rabbit, a technology originally developed in CERN. White Rabbit is based on layer 1 Ethernet physical synchronization and phase control of clocks directly communicating over highly symmetric fiber optic cables. This technique greatly reduces the noise floor. A calibration procedure is used to eliminate asymmetry introduced in the physical layer.

### *B. Accurate timing within a CPS node*

The second important requirement of time aware construction is achieving accurate timing within a node; i.e. inputs must be read and outputs written at or withing the times specified and the computations be completed in time to satisfy IO timing and latency requirements.

CPS designs typically use interrupt or polling to achieve temporal specifications. In polling-based techniques, sensing or actuating triggers are performed in a loop to sense/actuate at a fixed cadence. However, when polling multiple parameters in a loop, the loop must be carefully crafted to meet the required sensing frequencies. To do that, especially with ranging devices whose response latency varies depending on the environment (e.g., the distance of the objects), ensuring that all the sensing frequencies are correctly met becomes an art. Even if it is possible to design such a loop, any change in the code (software), or the devices used, completely changes the timing, and the whole analysis needs to be done all over again. In interrupt-based programming, a timer interrupt is set to trigger when a counter achieves a predetermined value. In the case of a system with multiple interrupts, priorities for interrupts are defined. An interrupt with higher priority can delay lower priority interrupts and cause a longer time interval than desired. Careful selection of priorities is required to meet the desired timing requirements. And even if possible, such design are typically brittle such that even small changes of software or hardware can result in completely different (and incorrect) behavior.




(a) Input devices (b) Output devices

Figure 3: Time aware architecture - (a) time stamp value is captured by clock and locked in the registers and (b) time stamp value of outgoing trigger is read from queue, locked in the registers and generated when clock is equal to time stamp.

In an effort to make programming of CPS a science (rather than an art), and reduce the brittleness of the application timing, we advocate the use of correct-by-construction design methodologies that guarantee the satisfaction of high level functional and timing specifications by the low level implementation. An important aspect of correct-by-construction design methodologies is to use time aware architectures [24]. Time aware architectures can record the exact moment of sensing, and can perform on-time actuation. When an event is detected in the physical layer, the event is time stamped with the current physical time in the hardware. Note that this physical time stamp may be translated to the global time base by the CPS. Figure 3a depicts a time aware architecture for incoming events. Since the time stamping of the incoming events is done in the hardware, the exact time of the event occurrence is preserved, and it not dependent on other hardware delays or the software schedule. Similarly, when the computation of an output is finished, it can be placed in the output queue together with the exact physical time when the actuation should happen. This time stamped value is then latched to be compared with the physical clock. When the clock time matches the latched time stamp, the actuator is executed (see Figure 3b).

Time aware architectures transform the problem of deterministic execution time, to the problem of worst case execution time analysis. This is because as long as the computation for the actuation finishes before the time stamp [25], the actuation event will happen at the right time. The actual execution time is not important since it does not influence the application behavior, but the execution time must fit within well defined

sensing and actuation schedules. In order to guarantee this property, the worst case execution time (WCET) of the computations on a given platform must be determined [26].

WCET analysis is inherently difficult as to some extent, in the chase for performance, computer architectures have developed to improve the average case performance of the processor, while trading off the time-predictability of the processor. The memory cache hierarchy is an excellent case in point. While caches improve the average case performance of applications by orders of magnitude, and therefore have become one of the basic pillars of modern computer architecture, they severely degrade the predictability of the execution time of a load/store operation. The time of the execution of a load or store operation can now vary by orders of magnitude, depending on where the data is found. While a lot of research has been done to estimate the WCET of tasks on a cache-based processors [26], but the techniques are hard to implement, provide very pessimistic WCET estimates, and work on only very simple programs, and have been developed mostly for instruction caches. It becomes even worse when multiple tasks are executing simultaneously on shared caches. We believe that the only way to obtain usable WCET estimations for complex large programs, we will need time-predictable architectures. The PRET machine [27] is one promising approach towards building architectures whose timing will be more predictable, but without losing the average-case performance. The two main features of PRET architectures are thread interleaving, so as to remove the pipeline effects on the execution time [28], and to use scratchpad memories, so as to achieve predictable execution time of load and store instructions [29]. It is easier to achieve tight estimation of execution time on scratchpad based processors, since in scratchpad based processors, the movement of data between the scratchpad and lower levels of memory has to be done explicitly in the program. The program must bring the data required into the scratchpad before it is needed. In fact every load/store instruction actually "hits" in the scratchpad (otherwise the execution would go wrong!). As a result, the timing analysis needs to be done for the programmer or compiler inserted data movement instructions, which is easier than the analysis of the data movement initiated by the hardware. Since scratchpad memories are just like caches, except that the movement of data is controlled by software (instead of hardware), equivalent performance can be expected pending only smart management of data. Recent compiler research on scratchpad memory based processors have shown that compilers can automatically manage code and data of applications very efficiently and can outperform caches [30], [31]. Further SPM can be partitioned among the multiple applications running simultaneously, so that they do not interfere with each other, and the execution time of each one of them remains predictable.

### *C. Managing latency in networks*

The third necessity of time aware designs is to manage the latency of communication. Time aware designs must enable correct timing in networks, in particular, the latency and synchronization requirements of the application must be satisfied.

In CPS, to achieve overall application performance and system stability, latency must typically be bounded, and in some cases also deterministic and small. Interfering best-effort traffic typically is the cause of PDV. In legacy LANs, one might consider elimination of interfering traffic, but doing so would be counter to the goal of the convergence of time-sensitive and best-effort traffic on the same network. These delay variations in the network not only make meeting the latency requirements difficult, but also pose challenges to correct synchronization. For example, most telecommunication networks are used continually at some level, and those levels continually change. In order to minimize the difference in the upstream and downstream PDV, all telecommunication PTP devices carefully select a tiny portion of the PTP packets for synchronization. This issue is not just in telecommunication – any Ethernet system with shared best effort and time sensitive traffic has the same problems [32]. Latency has the same problem with interference as synchronization packets, and many of the packets with latency requirements cannot be eliminated, as they are control words.

Many CPS are being designed for larger systems, some using public networks. On these systems, the time-sensitive traffic still has timing requirements, but it is merged with best-effort traffic. A CPS time domain is a logical group of computing nodes and bridges/routers which form a network with its own time master. To manage and configure a large CPS time domain, we suggest that one of the nodes should be configured to be the CPS Network Manager (CNM). This CNM is a software entity responsible for configuration and management of all CPS nodes in the CPS domain and interface with the network on their behalf. Time-sensitive data transfer occurs over one or more time-sensitive streams in the CPS domain. Each time-sensitive stream is transmitted by a single CPS node (the talker), and received by one or more CPS nodes (listeners). The CNM gathers time-sensitive stream requirements from all CPS nodes in its domain, and communicates with network entities to ensure that those stream requirements are met by the network. When the network is ready for time-sensitive data transfer, the CNM communicates stream transmission and reception information back to the CPS nodes (talkers and listeners). For example, the CNM tells each talker the scheduled time to transmit data for its stream. The functions of a CNM vary depending on the size of the system. A small CPS may not need a CNM, while a large system may have a distributed CNM, as is done for SDN. When dealing with multiple time domains as well as time masters, for example in a System of Systems, multiple CNMs may need to coordinate.

A Centralized Network Controller (CNC) is responsible for configuring the bridges/routers to meet the time-sensitive stream requirements. The CNM is responsible for users of the network (CPS nodes), and the CNC is responsible for the network itself. Figure 4 shows the configuration model for a CPS network.

It should also be noted that the functions of the CNM

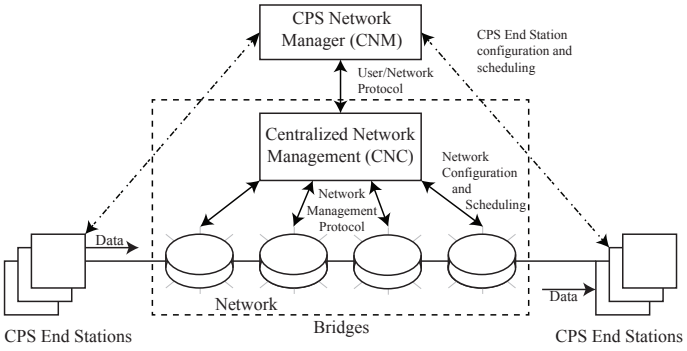



Figure 4: The CNM gathers requirements from nodes, or end stations, and determines schedules. It transmits these to the nodes, and passes the network schedules to the CNC, which configures the network nodes.

and CNC are being standardized in IEEE 802.1Qcc [33] as the Centralized User Configuration (CUC) and the Centralized Network Configuration (CNC). In addition, there is considerable activity in the IEEE 802.1 and other standards communities on enhancing standard networking to allow for implementation of the model suggested above by the IEEE 802.1 TSN TG "Time-Sensitive Networking Task Group". Other TSN enhancements to Ethernet networking include automatic duplication of packets for redundancy, Ethernet frame preemption, policing of misbehaving transmitters, a PTP Profile to propagate synchronized time and enable hardware time-stamping with redundancy and PDV-mitigation, an enhanced stream reservation protocol, network schedule configuration, and time aware frame transmission [34]. Depending on the clock synchronization protocol, additional network specifications on routing and selection of network components can be made, such as PTP aware clocks instead of ordinary switches. A means to mitigate PDV is the use of time aware network nodes such as PTP aware transparent clocks (TCs) and boundary clocks (BCs). They enable computation of packet delay based on *residence times* at each network node.

### *D. Virtualizing time without losing accuracy*

In general, a CPS node may be shared among several CPS applications. For example, in a smart city, a sensor on the street may be used for different purposes by different sets of people. They can all define their CPS, and recruit the CPS nodes to implement their functionality. In such a scenario, we will want to isolate the execution environment of one CPS application from others so as to provide security and even robustness from malicious actors. Virtualizing the CPS node is the most promising way going forward since it plays a significant role in hosting/managing distributed CPS. However, how can virtual machines get access to the desired clock source?

Every application potentially has a different understanding of the current time. These different clock domains must be comprehended within a VM without implementing multiple physical counters in the CPU. A scalable solution employs a single, free-running hardware counter in the CPU as well as a set of coefficients that represent the current linear relationship between the local CPU counter and the various PTP clocks. Any offset applied to the CPU counter per VM must be known by the VM and applied each time the CPU counter is read. A linear model, $y = mx + c$ is used to convert local CPU counter to remote PTP time, where $y$ is (or was) the time at the remote clock source corresponding to the time at the given CPU counter value $x$. The coefficients $m$ and $c$ are computed by periodically capturing simultaneously both the CPU counter and the relevant PTP time, which typically exists within the Ethernet or Wi-Fi device.

The requirements of scalable clock virtualization are fast multiplication and addition, immediate software access to a stable CPU counter, and precise estimation of $m$ and $c$. Given $m$ and $c$ for every PTP clock, software can convert a CPU time to any PTP clock and vice versa. When PTP is used in a virtualized environment, it is necessary to propagate event message time stamps to each guest OS that is running PTP, and to construct the appropriate residence time as though coming from a TC between the virtual network interfaces of the VMs. Figure 5 shows a schematic of such a system, with multiple virtual machines (VM) where PTP time is propagated through a virtual switch that includes TC functionality between the VMs.

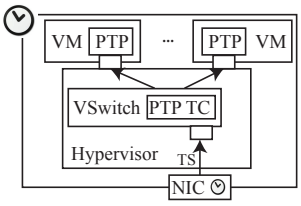



Figure 5: Virtualization of multiple PTP clocks for VMs - VSwitch implements PTP transparent clock to allow VMs individually interact with PTP hardware clocks

## VII. Testing the Timing

In order to test and validate that the final implementation satisfies the original application requirements, we propose a configurable timing testbed.

This testbed framework is complementary to tools such as simulators and emulators, as this is testing a physical realization of the application and its environment. All of these tools help improve the developers understanding of the proposed solution. The re-configurable testbed architecture is illustrated in Figure 6. It consists of the deployment target timed CPS nodes (A, B, C, D) and associated timed communication fabric for deployment of the cyber part of the application, a physical or hardware-in-the-loop (HIL)-based physics module, as well as physics monitoring and control target, where the generated test is deployed. All of these elements together form a closed system, i.e. a system where all the inputs are controlled and all the outputs are observed and measured in a well defined super-system. There are other support components, such as a supervisor interface that help coordinate the deployment of the application and the test, as well as a remote operator interface.

Every CPS node consists of a system stack with custom hardware at the bottom, interacting with the external physical world (to be measured or controlled) and the communication fabric. This layer is typically a combination of an FPGA, DACs, ADCs, communication physical layer (PHYs), input/output (I/O) etc., with standard computer interfaces to the operating system of the microprocessor. The designer creates either manually or by a correct-by-construction methodology code running on the microprocessor and a closely connected or integrated re-configurable fabric, which, in conjunction with the underlying hardware, realizes the functional and timing specifications of the CPS.

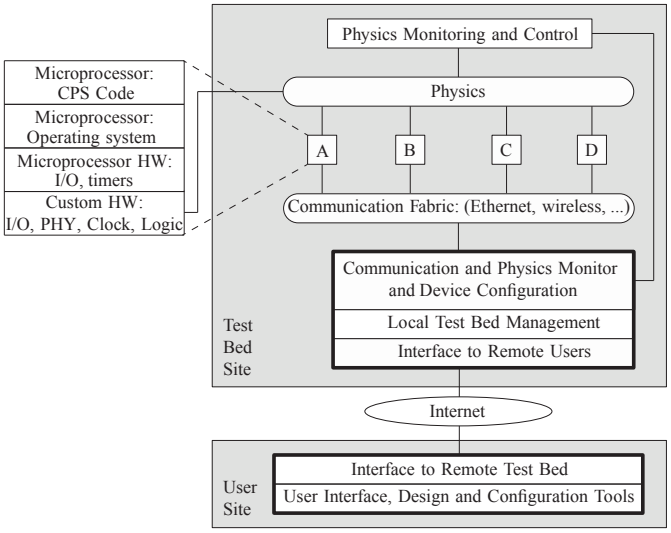



Figure 6: Testbed Platform Architecture, which provides a framework to help validate that an implementation satisfies the original application requirements

The testbed physics enables testing of time sensitive designs applied to realistic CPS applications and comparing it with alternative designs. The selection of components should be simple at first but at a minimum, should provide devices suitable for analog, digital and frequency dependent applications. Examples might include one or more of the following.

- Two small laboratory bench size motor-generator sets, mock transmission lines, capability to measure waveform properties like phase, and capability to connect/disconnect from a load. This could be used to mimic power system applications such as synchronizing two generators prior to connecting to a load.
- A digital pattern generator and capturing device to allow testing of stimulus response applications with sporadic or patterned signals.
- Two or more vibration sensors, perhaps mounted on the motor-generator to allow testing of machine condition and monitoring applications.
- The physics can be implemented in a Hardware-in-the-Loop (HIL) simulation, in which a powerful computer is simulating a real-time model of the physical plant.

The testbed monitoring capabilities depend on the specifics of the testbed physics. The monitoring generates a variety of time series data sets which can be fused together from local or remote locations. Ideally, the physics and monitoring is designed together and in many cases can be implemented using standard, small scale laboratory equipment available from several manufacturers. In some cases, this equipment can be synchronized to the PTP time scale to simplify comparison of ground truth monitoring with the results obtained from the CPS nodes. If the physics is simulated via HIL, physics monitoring is integrated into the model and connected directly to the rest of the system. Since the testbed is heavily dependent on the application requirements, we are exploring the automatic generation of parts of the testbed application from the requirements. We believe this can be done in a correct-by-construction way by using a platform model of the testbed hardware and, if possible, a model of the physics.

In addition to the application specification, it is useful to specify testbench information so a user can provide input stimuli at given times that are correlated to output responses at given times, and the testbed can be generated to check the validity of implemented testbenches. The input sequence is not necessarily limited to specific data, instead, it can be enhanced by pseudo-random sequences as well, with corresponding responses computed as per the functional specification. Similar work has been proposed as part of the Accellera's Portable Stimulus Working Group [1] for the digital IC design area.

The testbed facilitates the exploration of methodologies for re-using design information from design specifications, all the way to validation and production. It is important to use non-intrusive techniques for observations and measurements such that the behavior of the *System under Test* remains unchanged. A distributed testbed can bring together a community of multi-disciplinary experts to enable exploration and validation of time aware interfaces, correct-by-design methodologies, and measurement capabilities to a representative CPS architecture [35].

## VIII. Summary

The ability to reliably propagate a common physical time scale along with interfacing multiple time scales among observed physical dynamics and across platforms is crucial to correct and efficient functionality of complex, distributed CPS. This paper exposed the challenges in specifying the timing requirements of CPS, and the timing specifications of the CPS components. It further elaborated on the challenges in achieving synchronization among the CPS nodes, and the challenges in making CPS nodes perform actions at desired moments of time, managing delays in the networks, and virtualizing time without losing accuracy. The paper included a discussion on currently evolving formalisms, technologies and standards in realizing the vision of time aware CPS design and implementation. Finally, we presented a configurable testbed that can characterize and verify the temporal behavior of

[1] http://accellera.org/news/newsletters/2016-may

complex, distributed CPS based on correct-by-design methodologies and time aware platforms.

## ACKNOWLEDGMENTS

The authors would like to thank the NIST colleagues, Dr. Dhananjay Anand, Dr. Edward Griffor, Dr. Judah Levine, and John Messina, the NIST CPS Program for their support, the National Instruments colleagues Dr. Kaushik Ravindran and Dr. Arkadeb Ghosal, Dr. Hugh Melvin from National University of Ireland, Galway, and Mohammadreza Mehrabian from Arizona State University. This work was partially supported by funding from National Science Foundation grants CNS 1525855, and NIST grant 60NANB15D322.

## REFERENCES

[1] S. M. Amin and B. F. Wollenberg, "Toward A Smart Grid: Power Delivery for The 21st Century," *IEEE power and energy magazine*, vol. 3, no. 5, pp. 34–41, 2005.

[2] A. v. Meier and R. Arghandeh, "Every Moment Counts: Synchrophasors for Distribution Networks with Variable Resources," *arXiv preprint arXiv:1408.1736*, 2014.

[3] Y. H. Tang, G. N. Stenbakken, and A. Goldstein, "Calibration of Phasor Measurement Unit at NIST," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 6, pp. 1417–1422, 2013.

[4] J. Amelot, D. Anand, T. Nelson, G. Stenbakken, Y. S. Li-Baboud, and J. Moyne, "Towards Timely Intelligence in the Power Grid," in *44th Annual PTTI Meeting*, 2012.

[5] A. E. Motter, S. A. Myers, M. Anghel, and T. Nishikawa, "Spontaneous Synchrony in Power-Grid Networks," *Nature Physics*, vol. 9, no. 3, pp. 191–197, 2013.

[6] B. Liskov, "Practical Uses of Synchronized Clocks in Distributed Systems," *Distributed Computing*, vol. 6, no. 4, pp. 211–219, 1993.

[7] M. Toerngren, S. Tripakis, P. Derler, and E. A. Lee, "Design Contracts for Cyber-Physical Systems: Making Timing Assumptions Explicit," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-191*, 2012.

[8] P. Derler, J. C. Eidson, S. Goose, E. A. Lee, S. Matic, and M. Zimmer, "Using Ptides and Synchronized Clocks to Design Distributed Systems with Deterministic System Wide Timing," in *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*. IEEE, 2013, pp. 41–46.

[9] B. Kim, L. Feng, O. Sokolsky, and I. Lee, "Platform-Specific Code Generation from Platform-Independent Timed Models," in *Real-Time Systems Symposium, 2015 IEEE*. IEEE, 2015, pp. 75–86.

[10] K. Altisen and S. Tripakis, "Implementation of Timed Automata: An Issue of Semantics or Modeling?" in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2005, pp. 273–288.

[11] A. Sangiovanni-vincentelli, W. Damm, and R. Passerone, "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems," *European Journal of Control*, 2012.

[12] E. A. Lee, "The Past, Present and Future of Cyber-Physical Systems: A Focus on Models," *Sensors*, vol. 15, no. 3, pp. 4837–4869, 2015.

[13] O. Simeone, U. Spagnolini, Y. Bar-Ness, and S. H. Strogatz, "Distributed Synchronization in Wireless Networks," *IEEE Signal Processing Magazine*, vol. 25, no. 5, pp. 81–97, 2008.

[14] M. R. Mosavi and A. Tabatabaei, "Traveling-Wave Fault Location Techniques in Power System Based on Wavelet Analysis and Neural Network Using GPS Timing," *Wireless Personal Communications*, vol. 86, no. 2, pp. 835–850, 2016.

[15] C. Collaboration *et al.*, "Fine Synchronization of The CMS MUON Drift-Tube Local Trigger Using Cosmic Rays," *Journal of Instrumentation*, vol. 5, no. 03, p. T03004, 2010.

[16] D. Pilaud, N. Halbwachs, and J. Plaice, "LUSTRE: A Declarative Language for Programming Synchronous Systems," in *Proceedings of the 14th Annual ACM Symposium on Principles of Programming Languages (14th POPL 1987). ACM, New York, NY*, vol. 178, 1987, p. 188.

[17] E. Nunzi, L. Galleani, P. Tavella, and P. Carbone, "Detection of Anomalies in The Behavior of Atomic Clocks," *IEEE Transactions on Instrumentation and Measurement*, vol. 56, no. 2, pp. 523–528, 2007.

[18] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A Time-Triggered Language for Embedded Programming," *Proceedings of IEEE*, vol. 91, no. 1, pp. 84–99, 2003.

[19] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler, "Execution Strategies for Ptides, A Programming Model for Distributed Embedded Systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*. San Francisco, CA: IEEE, 2009.

[20] O. Maler and D. Nickovic, "Monitoring Temporal Properties of Continuous Signals," in *In: Proceedings of FORMATS-FTRTFT. Volume 3253 of LNCS*. Springer, 2004, pp. 152–166.

[21] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Transactions on communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[22] K. Lee, J. C. Eidson, H. Weibel, and D. Mohl, "IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in *Conference on IEEE*, vol. 1588, 2005, p. 2.

[23] J. C. Eidson, *Measurement, Control, and Communication using IEEE 1588*. Springer Science & Business Media, 2006.

[24] J. Eidson and K. Stanton, "Timing in Cyber-Physical Systems: The Last Inch Problem," in *Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, October 2015, pp. 19–24.

[25] J. Zou, S. Matic, E. A. Lee, T. H. Feng, and P. Derler, "Execution Strategies for PTIDES, A Programming Model for Distributed Embedded Systems," in *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*. IEEE, 2009, pp. 77–86.

[26] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-Case Execution-Time Problem ;Overview of Methods and Survey of Tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[27] S. A. Edwards and E. A. Lee, "The Case for the Precision Timed (PRET) Machine," in *Proceedings of the 44th annual conference on Design automation*. SESSION: Wild and crazy ideas (WACI), June 2007, pp. 264 – 265.

[28] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A Processor Platform for Mixed-Criticality Systems," in *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, April 2014.

[29] Y. Kim, D. Broman, J. Cai, and A. Shrivastaval, "WCET-Aware Dynamic Code Management on Scratchpads for Software-Managed Multicores," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2014, pp. 179–188.

[30] J. Lu, K. Bai, and A. Shrivastava, "SSDM: Smart Stack Data Management for Software Managed Multicores (SMMs)," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 149.

[31] K. Bai, J. Lu, A. Shrivastava, and B. Holton, "CMSM: An Efficient and Effective Code Management for Software Managed Multicores," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2013 International Conference on*. IEEE, 2013, pp. 1–9.

[32] NIST Public Working Group. (2015, September) Timing Framework for Cyber-Physical Systems, Technical Annex. [Online]. Available: https://pages.nist.gov/cpspwg/

[33] IEEE. (2016, June) 802.1Qcc - Stream Reservation Protocol (SRP). [Online]. Available: http://www.ieee802.org/1/pages/802.1cc.html

[34] IEEE. (2016, July) I. 802.1,Time-Sensitive Networking Task Group. [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[35] H. A. Andrade, P. Derler, J. C. Eidson, Y. S. Li-Baboud, A. Shrivastava, K. Stanton, and M. Weiss, "Towards a Reconfigurable Distributed Testbed to Enable Advanced Research and Development of Timing and Synchronization in Cyber-Physical Systems," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–6.