qPALS: Quality-Aware Synchrony Protocol for Distributed Real-Time Systems

Woochul Kang[1], and Lui Sha[2]
[1] School of Information Technology, Incheon National University
Incheon, South Korea
[e-mail: wchkang@incheon.ac.kr]
[2] Computer Science Department, University of Illinois at Urbana-Champaign
Urbana, IL, 61801, USA
[e-mail:lrs@illinois.edu]
*Corresponding author: Woochul Kang

Received June 7, 2014; revised August 12, 2014; accepted August 22, 2014; published October 31, 2014

Abstract

Synchronous computing models provided by real-time synchrony protocols, such as TTA [1] and PALS [2], greatly simplify the design, implementation, and verification of real-time distributed systems. However, their application to real systems has been limited since their assumptions on underlying systems are hard to satisfy. In particular, most previous real-time synchrony protocols hypothesize the existence of underlying fault tolerant real-time networks. This, however, might not be true in most soft real-time applications. In this paper, we propose a practical approach to a synchrony protocol, called Quality-Aware PALS (qPALS), which provides the benefits of a synchronous computing model in environments where no fault-tolerant real-time network is available. qPALS supports two flexible global synchronization protocols: one tailored for the performance and the other for the correctness of synchronization. Hence, applications can make a negotiation flexibly between performance and correctness. In qPALS, the Quality-of-Service (QoS) on synchronization and consistency is specified in a probabilistic manner, and the specified QoS is supported under dynamic and unpredictable network environments via a control-theoretic approach. Our simulation results show that qPALS supports highly reliable synchronization for critical events while still supporting the efficiency and performance even when the underlying network is not stable.

Keywords: Synchrony protocol, middleware, quality-of-service, QoS, cyber physical systems, real-time systems, feedback control, globally asynchronous locally synchronous, GALS

This work was supported by the Incheon National University Research Grant in 2014.

http://dx.doi.org/10.3837/tiis.2014.10.004

1. Introduction

Most cyber-physical systems (CPS) are distributed real-time systems having various scales of interacting entities in the cyber-physical world. One of the major challenges of such distributed systems lies in the complexity resulting from the asynchronous exchanges of data and control messages. The state-explosion problem due to the complex interleaving of data and control makes the systems hard to verify and, hence, vulnerable to potential safety hazards [3]. It has been well understood that synchronous computing models of distributed entities can significantly reduce the state-space and simplify the design, implementation and verification of distributed systems [4][5]. For this reason, synchronous models have gotten a great deal of attention as a promising paradigm to conquer the complexity of distributed systems [6][1]. Currently, however, the application of synchronous paradigms has been limited since synchronous paradigms assume all distributed entities are driven by a common clock, which is hard to achieve in loosely-coupled distributed systems. *Physically Asynchronous Logically Synchronous (PALS)* [2] has been proposed to make synchronous paradigms feasible in distributed systems in the absence of a common physical clock as long as several conditions and rules are observed. For example, PALS can guarantee (virtual) global synchrony as long as the worst-case local clock skews, task execution time, and network latency are bounded. These are feasible properties in some systems such as avionics flight control systems that have fault-tolerant and real-time communication mechanisms [2].

However, unfortunately, synchrony protocols including PALS still manifest challenges and problems when applied to most CPS. In particular, for most CPS end-to-end network latency is dynamic and it is hard, if not impossible, to provide a firm worst-case bound without significantly compromising the performance. Further, since the operating environment of CPS are highly dynamic, a synchrony protocol for CPS needs to provide a flexible mechanism to control the level of synchronization and its overhead in a predictable manner. For instance, in an integrated medical environment [7], the synchronized feeds from medical sensors can give better understanding of a patient's state, e.g., cause and effect relation between events, but missing a few events are still tolerable as far as the ratio is bounded and controlled. However, in the same system, some medical devices, e.g., oxygen ventilator and airway laser, have interlocks and should be coordinated in synchronous manner to guarantee the safety of the surgery [8]. In this situation, the synchronization of the interlocking devices should be highly reliable. As this example shows, many CPS require a synchrony protocol that can meet these different synchronization requirements efficiently and in a controlled manner.

In this paper, we propose a synchrony protocol, called *Quality-Aware PALS*, or *qPALS*, to support flexible synchronous computing for dynamic CPS. In qPALS, the Quality-of-Service (QoS) is quantified as the ratio of successful synchronization. The primary goal of qPALS is to make the synchrony protocol of PALS more adaptable and flexible, rendering asynchronous distributed systems to take advantage of the synchronous paradigm in a computing environment where firm real-time guarantees are not easily achievable.

For flexibility, qPALS supports two synchronization protocols: *p-synchrony* and *c-synchrony*. In *p-synchrony*, which is tailored for performance, individual incidents of synchrony violation are not reported to the applications, but only the ratio of synchrony violations is known to the applications. Therefore, a task using *p-synchrony* class should handle synchrony violations using application-specific semantics. In contrast, for applications, in which each incident of synchrony violation is critical for maintaining the consistency,

c-synchrony protocol provides a group management protocol. The group management scheme of *c-synchrony* provides the consistency among group members by excluding a member process as soon as it violates synchrony and potentially breaks the consistency. The desired QoS of *c-synchrony* is enforced by exploiting spatial and temporal redundancy of communication messages, which incurs additional overheads for synchronization. Since *p-synchrony* has a low communication overhead and latency, it is appropriate for exchanging real-time sensor readings, which has inherent uncertainty. In contrast, *c-synchrony* of qPALS is appropriate for exchanging critical control messages. By composing the two proposed synchrony protocols together in one synchronization group, the group can support highly assured consistency while still maintaining the efficiency. Further, qPALS provides the adaptiveness and robustness against dynamic network environments to support the specified QoS of synchronization. Previous synchrony protocols cannot provide synchronization under unstable networks since most of them have been designed and implemented assuming firm real-time guarantees. To address this problem, qPALS takes a controlt-heoretic approach to providing the robustness against unpredictable networks. qPALS adjusts its parameters autonomously to support the specified level of synchronization.

Through a simulation study, we show that qPALS supports a highly-assured synchronization, while still supporting efficient synchronization for frequent, but less strictly-synchronized events. Our simulation results also demonstrate that qPALS can maintain these desirable properties even when the underlying network is not stable.

The rest of this paper is organized as follows. In Section 2, we place qPALS in context by examinging relevant works. Section3 presents the background on the PALS synchrony protocol. The details of qPALS protocol is discussed in Section 4. In Section 5, we present the simulation results in the context of an example application. Section 6 concludes the paper and discusses future work.

2. Related Work

Distributed middleware, such as real-time CORBA [15] and Data Distriubtion Service (DDS) [13], provide a virtualized platform for distributed tasks to collaborate. However, with these middleware, the developers should be aware of the asynchronous nature of the distributed interactions. In contrast, the middleware based on PALS protocol provides a synchronous computing platform that hides the physical asynchrony and simplifies the distributed algorithms. PALSware [11] is one of such middewares based on PALS protocol. In this paper, we extend PALSware to support PALS protocol even if the environment does not provide fault-tolerant real-time communication.

Providing distributed processes with consistent views has been actively investigated in the general-purpose computing [20]. For example, Pereira et al. [24] proposed a low-cost virtual synchrony, in which a group management protocol is integrated into the multicast subsystem. However, these virtual synchrony approaches focus on ensuring consistent views of the global state without the notion of physical time. In contrast, real-time synchrony protocols such as qPALS exploit the existence of a global system clock and have a firm timing bound of synchronization.

Synchronous models of computation have been applied in most digital circuit designs due to the complexity of asynchronous abstraction. In software, there have emerged various synchronous programming languages, such as Esterel [4] and PRET-C [5], to take advantages of the model's simplicity [21]. They assume that there exist underlying frameworks to support

synchronous semantics. For instance, PRET-C extends a general processor with a hardware accelerator for predictable execution of synchronous tasks. In qPALS, unlike PRET-C, software middleware is responsible for supporting predictable execution of distributed synchronous tasks. In this paper, we propose a middleware-based approach to predictable execution of distributed synchronous tasks.

Time Triggered Architecture (TTA) [1] is one of the earliest system architectures that introduced distributed real-time clock sources and has been actively investigated for industrial applications [18]. Both TTA and PALS have been designed for safety-critical systems, and, hence, they are both formally verified [19]. TTA achieves global state synchronization by introducing a single global clock implemented in the system bus. Unlike TTA, PALS achieves global synchrony without specific hardware support as long as the PALS protocol assumptions, such as the communication bound μ_{max}, are satisfied. The schemes of qPALS, introduced in this paper, make these PALS assumptions more feasible in environments where a fault-tolerant real-time network is not available.

Probabilistic approaches to reliable multicasts and broadcasts have been considered as a viable alternative to traditional deterministic reliable approaches [22][23]. However, the primary goal of these probabilistic approaches has been in achieving a higher scalability in large-scale distributed systems. In contrast, the probabilistic approach of qPALS is to provide reliable synchronization, not scalability, with real-time guarantees.

3. The PALS Protocol Background

This section introduces original PALS protocol, and discusses the assumptions and constraints that must be satisfied to provide the global synchrony for cooperating distributed asynchronous entities. These assumptions are relaxed in the following sections on qPALS.

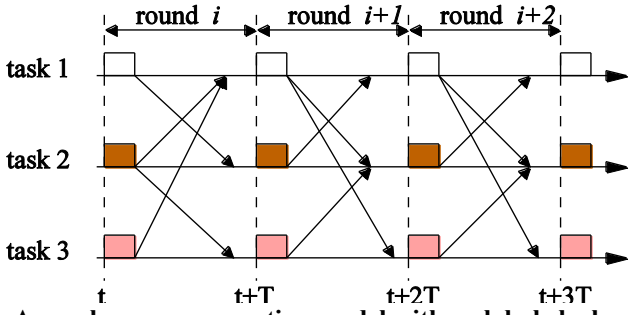


Fig. 1. A synchronous computing model with a global clock source.

Fig. 1 depicts a synchronous model of computation. In the synchronous model of computation, distributed nodes perform computation and communicate in coordination according to common clock ticks. In the model, the computation is chunked into series of rounds in time. In each PALS round, computation of every node is performed simultaneously, hence computation of one node cannot influence another node's at the same round. After a node finishes its computation at a round, it packages the result into packets and distributes them to other nodes. The messages are delivered and used as inputs at the next round. In most implementations of the model, such as digital circuits, the distributed computation is triggered by a global clock source, which generates monotonical increasing clock ticks.

PALS protocol is designed for software systems that cannot employ a physical global clock source. Instead, the distributed computation in the PALS protocol is initiated by the local

clocks at each node. The following are assumed about the environment for PALS protocol:

1) *Maximum clock skew* ε of a local clock is bounded with respect to a global clock.
2) *Network communication latency* μ is bounded, i.e. $\mu_{min} \le \mu \le \mu_{max}$.
3) *Computation time* α at each node is bounded, i.e., $\alpha_{min} \le \alpha \le \alpha_{max}$.

Since local clocks are not perfectly synchronized, the following two PALS protocol rules must be satisfied to guarantee the synchronization among the participating nodes:

R1: PALS clock period - PALS period *T* should be long enough, as shown in Equation 1, to ensure that a message sent in round *i* by a node arrive in round *i* of different nodes. The arrived message is consumed by the different nodes in round *i* + 1.

$$T \ge \mu_{max} + 2\varepsilon + max(\alpha_{max}, 2\varepsilon - \mu_{max}) \quad (1)$$

R2: PALS causality - To prevent a message from arriving too early, a node should have a minimal delay *H* after a PALS clock tick before sending a message.

$$H \ge max(2\varepsilon - \mu_{\min}, 0) \quad (2)$$

Meseguer et al. formally proved that the PALS system preserves the logical equivalence with a globally-clocked synchronous system as long as above assumptions and rules are observed [9]. However, the 2nd assumption on the communication bound μ_{max} is feasible only if a fault-tolerant real-time network is available. In this paper, we assume a network environment, in which the communication bound μ_{max} is not deterministic but can be described only probabilistically. Accordingly, qPALS assumes the following characteristics of the underlying communication network:

Failure model: Messages can be lost for any reason, e.g., transient link failures. Communication links do not have firm communication bounds. In qPALS, messages with latency longer than μ_{max} are considered being lost. Nodes, or tasks fail by crashing. Task failures are detected only when a message loss is detected. For the detection of task failures, qPALS can be combined with preferred mechanisms of detecting task failures [10]. The implementation details to detect node failures using end-markers is discussed in [11].

4. Quality-Aware PALS Protocol

4.1 Overview

The primary goal of qPALS is to provide flexibility and adaptability to the original PALS protocol. To this end, qPALS supports two distinctive synchronization protocols: *p-synchrony* and *c-synchrony*. The two synchronization protocols can be composed into a single synchronization group while preserving the benefits of the synchronous computing model.

p-synchrony is a basic building block, in which the communication latency bound μ_{max} and according PALS period *T* are determined to meet the QoS requirements. In this paper, we define *synchronization ratio (SR)* as the primary QoS metric of qPALS. The synchronization

ratio is defined as follows:

$$SR = 100 \times \frac{\#Shnchronized}{\#Synchronized + \#Violated}(\%), \tag{3}$$

where *#Synchronized* and *#Violated* represent the number of synchronization rounds that synchronized and violated the communication bound μ_{max}, respectively. For instance, if any message misses μ_{max}, the synchrony of that round is violated. In *p-synchrony*, the specified synchronization ratio is supported, but individual violations of synchrony are not reported to the applications. Hence, the synchronization using *p-synchrony* is probabilistic. The *adaption manager* of *p-synchrony* is a key component, which is responsible for maintaining the specified synchronization ratio under unstable networks. A control-theoretic approach, which is lightweight in computation, is taken to address this problem.

c-synchrony is built on top of the *p-synchrony* to provide a higher level of assurance on the synchronization and consistency. Unlike *p-synchrony*, *c-synchrony* maintains a synchronization group, in which only synchronized and consistent member entities are included. The reliable multicast exploits the redundancy to make all members of the synchronization group to maintain consistent states in a synchronous manner. Omissions of messages and potential inconsistency are detected during the reliable multicast, and the group is maintained to keep the consistency among the members. A synchrony round of *c-synchrony* is called a *hyper cycle* that is composed of a few successive *p-synchrony* rounds. The length of the hyper cycles is determined by the desired QoS level of applications.

4.2 P-synchrony

As discussed in Section 3, the benefits of synchronous computing model are preserved as long as the constraints of PALS protocol are observed. In particular, we are interested in guaranteeing the communication bound μ_{max} in non-fault tolerant and non-real-time networks. When the probability of breaking the communication bound μ_{max} is given as follows

$$P(\mu \geq \mu_{max}) \leq p, \tag{4}$$

the QoS level of the *p-synchrony* can be stated as follows:

$$SR_{psync} = (1-p)^n \times 100(\%), \tag{5}$$

where n is the number of tasks, or nodes, in the synchronization group. For example, if $p = 10^{-3}$ and n = 10, approximately 1 out of 100 synchronous state transitions fails because several messages cannot not arrive within the communication bound μ_{max}. We can increase the QoS level by setting a longer communication bound μ_{max}, and PALS period T, but increasing SR compromises the performance of synchronization. Since the rate of synchronous communication should be decreased. If highly assured synchronizations are required infrequently, using *c-synchrony* is desired, rather than increasing the communication bound of *p-synchrony*. It should be noted that this probabilistic and loose synchronization is sufficient in many distributed computing jobs of CPS [12]. For instance, in many CPS, sensors have inherent uncertainties and the freshness of sensor data is more important than exact synchronization. Since the semantics of a physical situation observed by sensors can be exploited by the applications to tolerate the lost messages as far as the level of message loss and delay is bounded. For example, **Fig. 2** shows that sensor readings are replicated to multiple tasks, and each task changes its state accordingly. During the state transitions,

Kalman filters are used to overcome the inherent uncertainty of sensor measurements and communications [13]. In the synchronous computing model of original PALS, the message omission at the task_2 in the 2nd PALS round breaks the synchronization and consistency. However, since missed sensor readings can be estimated locally from task_2's filter, the consistency among the tasks can still be maintained with high accuracy. Readers are referred to [14][13] for more information on how broken synchronization can be handled using application-specific semantics.

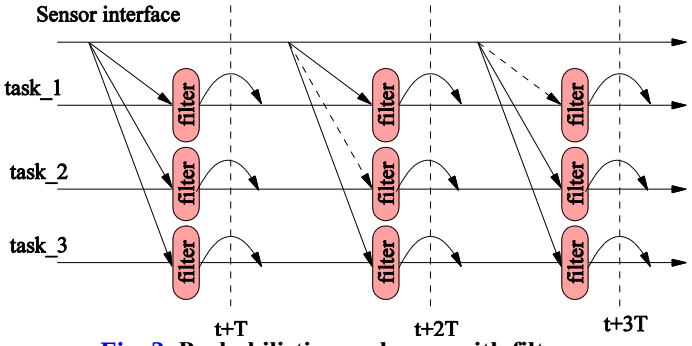


Fig. 2. Probabilistic synchrony with filters.

4.2.1 Robustness under Unstable Networks

The *p-synchrony* depends on the assumption that the probability p in Equation 4 is stable. However, in many networking environments, p typically manifests instability. Without considering those instability of p, the QoS of synchronization given in SR_{psync} and SR_{csync} cannot be supported. Therefore, the adaptation manager of qPALS is responsible for adjusting μ_{max}, and according T, dynamically at runtime to assure that Equation 4 remains valid. A statistical approach might be considered, but statistical approaches that need extensive historical data are less applicable to CPS due to potential runtime overheads. Instead, we take a control-theoretic approach to maintaining the validity of the original probabilistic assumptions of qPALS.

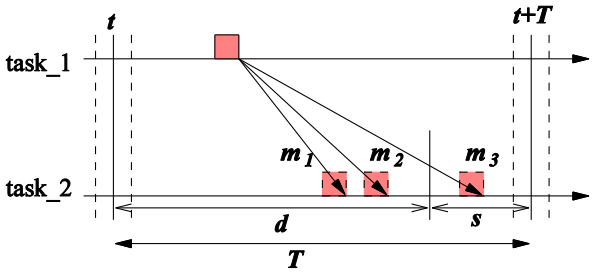


Fig. 3. Pseudo-deadline *d* and slack time *s*.

First, as shown in **Fig. 3**, while we set initial μ_{max} and T in Equation 4, we also set a pseudo (relative) deadline d such that the following is satisfied:

$$P(\mu \geq d) \leq p_{pd}, \tag{6}$$

where $d = c \times T$ ($c < 1$; c is a constant) and p_{pd} is a target probability, e.g., 0.9. The

pseudo-deadline *d* is not a real deadline, but it is to probe the changes in the communication latency. For instance, the message m_3 in Fig. 3 missed the pseudo-deadline *d*, but still satisfies the real communication bound μ_{max}. Intuitively, if the ratio of messages violating the pseudo-deadline, such as m_3 in **Fig. 3**, is greater than p_{pd}, the current μ_{max} and T are too short to guarantee the initially made probabilistic guarantees. Conversely, if the ratio of messages arriving earlier than the pseudo-deadline is greater than $1 - p_{pd}$, then this indicates that the current μ_{max}, and *T*, is too long. At runtime, to measure these changes in the communication latency, we define deadline miss ratio *MR* as follows:

$$MR = \frac{\#m_{tardy}}{\#m_{tardy} + \#m_{timely}} \times 100(\%), \tag{7}$$

where $\#m_{timely}$ and $\#m_{tardy}$ are the total number of timely messages and tardy messages (with respect to the pseudo-deadline), respectively. By maintaining MR close to MR_{target} (= p_{pd} ×100(%)), we can guarantee the validity of the probabilistic assumption in Equation 4. The overall feedback control procedure is as follows:

1) At each task, $\#m_{timely}$ and $\#m_{tardy}$ are measured on every monitoring cycle, and MR is computed.
2) Based on $|MR - MR_{target}|$, the adaptation manager of qPALS computes the control signal $\Delta\mu_{max}$.
3) each task unicasts its $\Delta\mu_{max}$ to a designated coordinator task. The coordinator conservatively takes the maximum of them, MAX($\Delta\mu_{max}$).
4) $\mu'_{max} = \mu_{max} + MAX(\Delta\mu_{max})$ is a new PALS communication bound. The coordinator (reliably) multicasts μ'_{max} to group members.
5) new PALS period *T* and pseudo-deadline *d* are calculated using μ'_{max} and it is applied in all group members from the next PALS round aligned to the next hyper cycle.

One valid criticism of the presented adaption algorithm is that it treats message losses and delays equally. Our assumption here is that packet losses and delays are highly correlated, and, hence, controlling one can also effectively control the other. However, this may not always true. In such situation, we should consider using more reliable transport mechanisms. With a reliable transport like TCP, we can effectively convert packet losses to end-to-end communication delays.

4.3 C-synchrony

In *c-synchrony*, the consistency among distributed entities is maintained by the group management protocol. For example, a new PALS period *T* determined by the adaptation manager is multicast using *c-synchrony*. Unlike *p-synchrony*, a member node of a synchronization group is excluded from the group if it does not get a multicast message. To this end, synchronous communication of *c-synchrony* occurs in 2-phases, called *hyper cycle*. In the 1st phase, called MULTICAST phase, messages are multicast to the group members. During the MULTICAST phase, inconsistent tasks, who lost the multicast message, are marked. In the following MEMBERSHIP_CHANGE phase, the inconsistent tasks, who lost

multicast messages, are excluded from the group. **Fig. 4** shows an example of one hyper cycle, in which 6 PALS rounds are performed.

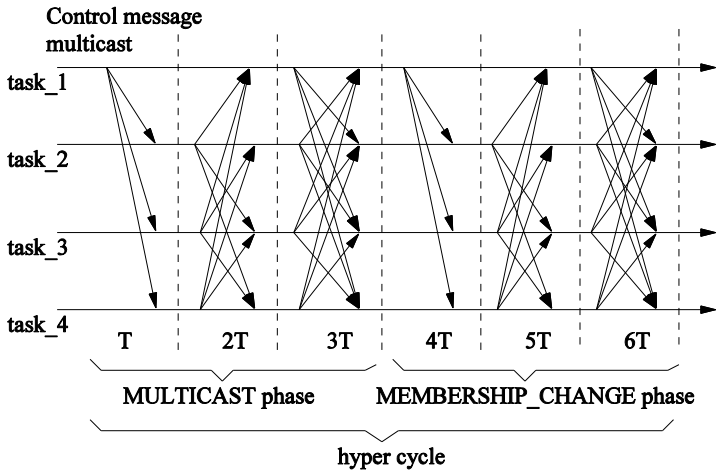


Fig. 4. Two phases of *c-synchrony*.

The propagation of a multicast message at the MULTICAST phase is similar to gossip-based protocols [16]. **Algorithm 1** shows the message handler at the MULTICAST phase, which is invoked at each PALS round. Two parameters f and k determine the assurance level and the latency of the reliable multicast; f is the fan-out and k is the number of PALS rounds. **Fig. 4** shows an example of a reliable multicast with $f = 3$ and $k = 3$. A control message from task *task_1* is propagated to other group members. The redundancy level at each task after k rounds is given as follows:

$$\text{redundancy level} = (f \times (k - log_f n) + 1) \tag{8}$$

Equation 8 indicates that each task gets the same message $(f \times (k - log_f n) + 1)$ times in average during the hyper cycle. Hence, the synchronization ratio of *c-synchrony* with n tasks is estimated as follows:

$$\text{SR}_{csync} = (1 - p^{f \times (k - log_f n) + 1})^n \times 100(\%), \tag{9}$$

where p is the probability given in Equation 4. As we increase k, the higher probabilistic assurance on correct synchronization can be achieved. However, it should be noted that the cost also increases accordingly. For example, the latency of c-synchronization is proportional to PALS rounds, k. Since the maximum number of messages in *inQueue* at line 1 is $m(n-1)$, the time complex of **Algorithm 1** becomes $O(m(n-1))$, where m is the number of distinctive messages at the PALS round and n is the number of group members. Since the multicast handler in **Algorithm 1** is invoked once at each PALS round, the total computational complexity of MULTICAST phase is $O(k \cdot m(n-1))$.

Group management of qPALS is bundled with each *c-synchrony* multicast, and it tells which tasks have lost multicast messages, and are not consistent with other tasks in the group.

```
   Input: inQueue: local queue of received multicast messages
   Input: haveSeen[m]: a set of members who have seen message m
   Output: outBuffer: local buffer of messages to send
1  foreach msg in inQueue do
2      haveSeen[msg.id] ← haveSeen[msg.id] ∪ msg.haveSeen;
3      if outBuffer[msg.id] ≠ NULL then
4          outBuffer[msg.id] ← msg;
5      else
6          outBuffer[msg.id].haveSeen ← haveSeen[msg.id];
7      end
8  end
9  foreach msg in outBuffer do
10     multicast msg ;
11 end
```

Algorithm 1. *c-synchrony* MULTICAST phase handler.

```
   Input: inQueue: local queue of received multicast messages
   Input: liveProc: local list of live group members
1  foreach message msg in inQueue do
2      liveProc ← liveProc ∪ msg.liveProc;
3  end
4  multicast liveProc;
```

Algorithm 2. *c-synchrony* MEMBERSHIP_CHANGE phase handler.

To detect failed tasks, each multicast message *msg* at the MULTICAST phase is appended with information *msg.haveSeen* that tells who have seen the message. Further, each task also maintains a *haveSeen* list for each message. Hence, when a multicast message is received by a task, the task updates its *msg.haveSeen* list and also updates the message's *haveSeen* list before replicating the message to other tasks. For example, when a message with *haveSeen* = {task_0, task_1} is delivered from task_0 to task_2 with *have_seen* = {task_2, task_3}, the task_2 updates its *haveSeen* list to {task_0, task_1, task_2, task_3} and modifies the message's *haveSeen* list too before sending it out. After the MULTICAST phase, each task can determine if any task in the synchronization group have missed the multicast message by checking its *haveSeen* list. At the MEMBERSHIP_CHANGE phase, each task multicast its *haveSeen* list to other memers. **Algorithms 2** shows the message handler at the MEMBERSHIP_CHANGE phase, which is invked once at each PALS round. Since the maximum number of message in *inQueue* at line 1 is *n-1*, the total time complexity of the handler is *O(n-1)*. Further, since the hander is invoked once at each PALS round, the total computational complexity of MEMBERSHIP_CHANGE is *O(k(n-1))*.

4.4 Composition of P- and C-synchrony

In qPALS, we have two synchrony protocols operating in different timing boundaries. In our earlier work, we showed that the composition of two proposed synchrony protocols, which are operating in different timing boundaries, can be done without breaking the semantics of the original synchronization [17]. When two synchrony protocols are composed in a single synchronization group, their interaction should be coordinated to ensure that the intended

consistency is not broken. To this end, the following rule is added to the original PALS protocol:

R3: Composition - A message delivered by a multicast of *c-synchrony* is visible to other group members at the final PALS round of the hyper cycle.

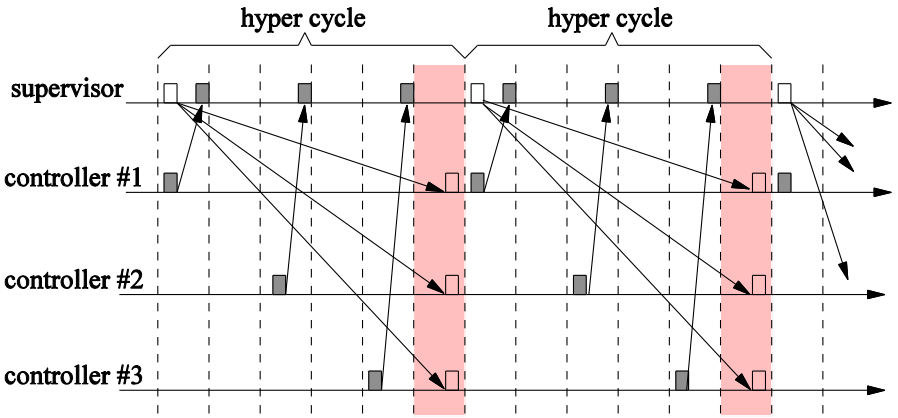


Fig. 5. Composition of p- and *c-synchrony*.

As an example, consider a hierarchical control system in **Fig. 5**, where a supervisor controller controls 3 underlying controllers by multicasting its commands periodically. The underlying controllers also periodically report their status to the supervisor, which in turn makes a higher level control decisions at the next hyper cycle. The supervisor controller performs a multicast in every *c-synchrony* hyper cycle, and the underlying controllers report their status with *p-synchrony*. In **Fig. 5**, even though the actual command from the supervisor is delivered to the controllers in the 1st or 2nd PALS rounds of the *c-synchrony* hyper cycles, and replicated several times during the hyper cycle, they are not visible to the controllers until the final round of the hyper cycle; hence, **R3** is enforced. This rule makes the controllers #1-#3 react to the command from the supervisor synchronously. Otherwise, some controllers react early, and the others react lately, making subtle inconsistency.

5. Evaluation

In this section, we use an active standby architecture shown in **Fig. 6** as an example to show the performance and correctness of synchronization in qPALS. The active standby system has two physically separated controllers, *controller_1* and *controller_2*. These two controllers are analytically redundant, and, hence, they are not exact replicas to each other. The *sensor input synchronizer* multicasts a stream of sensor data to both controllers. Only one of them is active while the other stays standby. The *supervisory controller* multicasts commands such as 'switching the active controller'. The commands from the supervisory controller should be strictly synchronized. Otherwise, non-deterministic interleaving of sensing and control on asynchronous architecture can create critical problems such as deadlocks and race conditions. Miller et. al. [2] showed that synchrony protocols such as PALS can significantly, e.g., several orders of magnitude, simplify the verification of the system given in this example. This significant reduction of complexity is possible only if the assumed synchronization is guaranteed. In the following simulations, we demonstrate that qPALS can effectively guarantee the synchronization without significantly compromising performance.

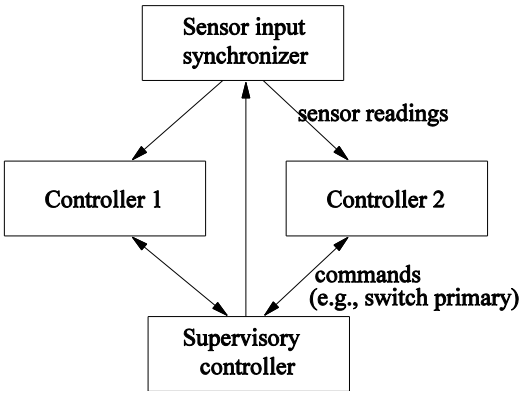


Fig. 6. Active standby architecture.

Table 1. Tested Approaches.

Approaches	Explanation
O-PALS	Original PALS with a fixed communication bound (μ_{max})
qPALS	PALS supporting dynamic communication bounds and flexible synchrony protocols

5.1 Baselines and Evaluation Goals

In the simulation, we compare our approach, qPALS, with the original PALS protocol, *O-PALS*. O-PALS has a fixed communication bound, μ_{max}, while qPALS supports both dynamic adaptation of μ_{max} and the composition of *p-* and *c-synchrony* protocols.

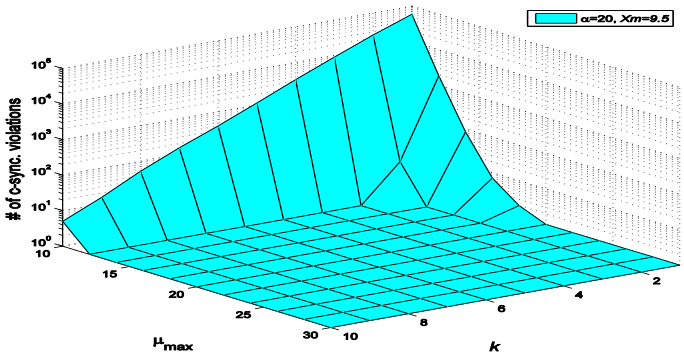
The two objectives of the evaluation are 1) to assess the effectiveness of *p-synchrony* of qPALS under diverse communication environments and 2) to test if qPALS can still meet the QoS specification when the network environment is not stable. For the first objective, in Experiment #1, we investigate how many synchronizations are violated while the configuration parameters of the simulation is varied. In experiment #1, the adaptation manager of qPALS is disabled to test the effectiveness of *c-synchrony*'s redundant multicast mechanism alone. For the second objective, in Experiment #2, we investigate the transient behavior of qPALS while the communication environment is changed dynamically during the execution. In this experiment, the adaptation manager is activated for qPALS.

5.2 Experiment #1: Average Synchronization Performance

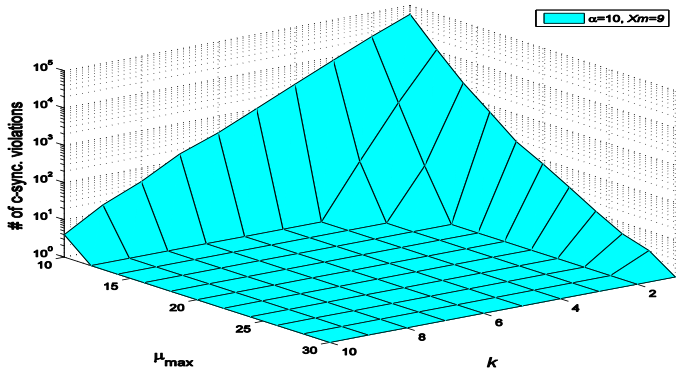
In this experiment, the supervisor send commands periodically to the two controllers on every 200*ms* and the number of synchronization violations is observed under varying communication environments. In qPALS, the commands from the supervisory controller are multicast redundantly using *c-synchrony* while the level of redundancy *k* is varied. When *k*=1, qPALS and O-PALS are equivalent. Commands are sent 100,000 times from the supervisor. The violated synchronization means that either commands from the supervisory controller is lost or delayed during the multicast. In such situation, the correctness and the safety of the system cannot be guaranteed. For example, one controller can become active while the other is still active. Such situation must not happen in an ideal synchronous computing model.

In the simulation, the communication latency is assumed to follow Pareto distribution with the mean latency of 10*ms*. We simulate two representative network environments by assigning two different sets of parameters. In the first configuration, the communication latency distribution's shape parameter α and the minimum parameter *Xm* are set to 20 and 9.5, respectively. This configuration represents communication networks with very low variances in latency. In the next configuration, we set α=10, rendering a longer tail in the communication latency. This configuration represents non-real-time network with larger variances in communication latency.

Fig. 7-(a) shows the result when α=20 and *Xm*=9.5. The z-axis shows the number of synchronization violoations out of 100,000 commands while k and μ_{max} are varied. When k=1, zero synchronization violations is achieved only at μ_{max}=20*ms*. In contrast, when k=3 zero synchronization violations is achieved at μ_{max}= 12*ms*. This demonstrate that qPALS can achieve reliable synchronization performance with shorter μ_{max}. For example, qPALS with k=3 can operate maximally at about 80Hz (=1 *sec*/12 *ms*) for sensor data communicaiton while still achieving reliable communication using *c-synchrony*. In contrast, in O-PALS, both sensor data and commands are delivered at the same rate, which is about 50Hz (=1 *sec*/20*ms*), to achieve reliable syncrhonization.



(a) α = 20, Xm=9.5

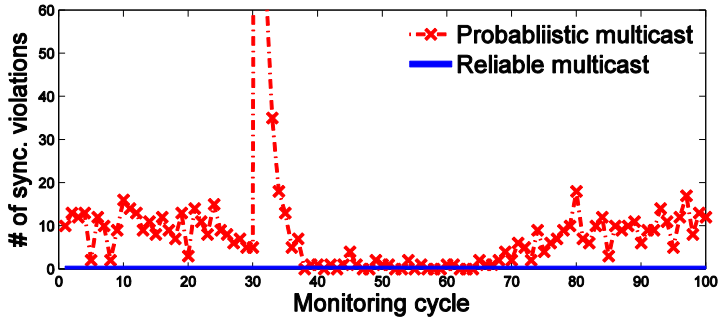


(b) α = 10, Xm=9

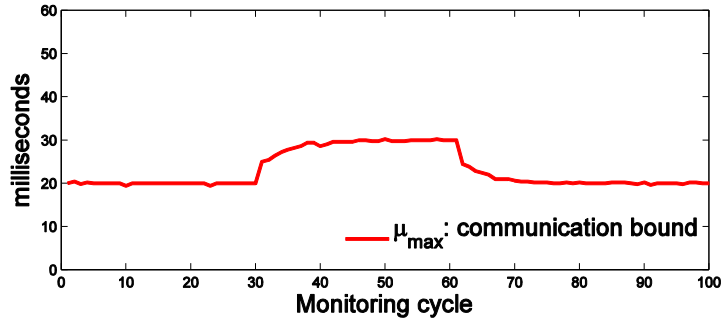
Fig. 7. Number of sychronization violations.

As the distribution of communicatino latency has longer tail, this performance gap becomes more evident. **Fig. 7-(b)** shows the result when α =10 and *Xm*=9. In **Fig. 7-(b)**, when *k*=1, zero synchronization violations is achieved only when μ_{max} is greater than 30ms. In contrast, when *k*=4, *c-synchrony* of qPALS can achieve zero synchronization violations at $\mu_{max} = 12ms$. This means that, for reliable synchronization, qPALS can still operate maximally at 80Hz while O-PALS needs to operate at 33Hz (=1*sec*/30*ms*).

5.3 Experiment #2: Transient Performance



(a) Synchronization violations



(b) Communication bound (μ_{max})

Fig. 8. Transient behavior of qPALS.

In this experiment, we observe the transient behavior of both qPALS and O-PALS for 100 monitoring periods, and suddenly increase the average communication latency to test the robustness of qPALS against unstable network environments. In the beginning, the communication latency follows Pareto distribution with α=20 and *Xm*=9.5. At 30*th* monitoring period, we intentionally increase the average communication latency by 50% for the next 30 periods. Initially, the communication bounds, μ_{max}, of qPALS and O-PALS are set to 20*ms* and 60*ms*, respectively. We give a longer μ_{max} to O-PALS since it cannot make reliable synchronization at μ_{max}= 20ms. One monitoring interval corresponds to 10,000 PALS periods and we set the redundancy level *k* of *c-synchrony* to 3.

Fig. 8 shows the transient behavior of qPALS for 100 monitoring cycles. In **Fig. 8-(a),** the number of synchronization violation in the *p-synchrony* multicasts is depicted in y-axis. In **Fig. 8-(b)**, the changes of the communication bound μ_{max} is shown in the same experiment. Until

30*th* monitoring period, the communication bound μ_{max} is quite stable, staying at 20*ms*. This indicates that the ratio of messages taking longer than the pseudo-deadline *d* remains almost constant at each monitoring cycle. Hence, the adaptation manager of qPALS is not actively involved. At the 30*th* monitoring period, the network latency changes suddenly, and, in consequence, a significant number of probabilistic multicasts violate synchronization. However, the number of violations drops and stabilizes within 10 monitoring cycles. This is because the adaptation manager of qPALS is actively involved and adjusted μ_{max} adaptively to around 30*ms* as shown in **Fig. 8-(b)**.

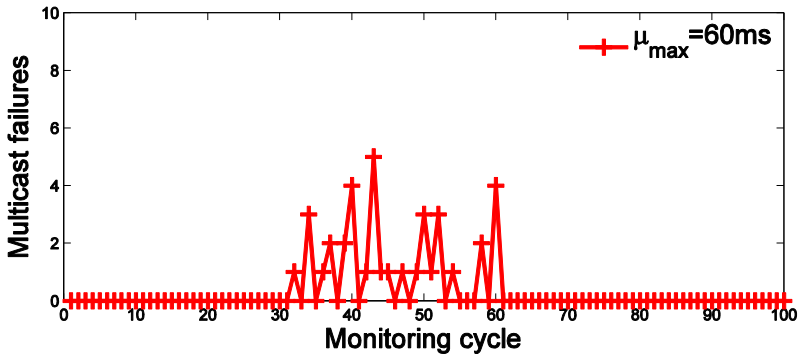


Fig. 9. Transient behavior of O-PALS (μ_{max}=60ms).

It should be noted that the reliable multicasts of qPALS using *c-synchrony* does not suffer from synchrony violations even when the network latency changes suddenly. **Fig. 8-(a)** shows that the number of synchronization violations in the reliable multicasts remains zero during this upheaval. This result demonstrates that qPALS is highly robust against the network changes by combining the redundancy-based reliable synchrony protocol and the adaptive communication bounds.

Fig. 9 shows the result when the same experiment was performed for O-PALS. The result shows that O-PALS cannot adapt to the changes of the network latency. O-PALS breaks the synchronization several times during the unexpected surge of the network latency. Again, these synchronization failures can be a source of subtle problems such as race conditions. Hence, the integrity of the system cannot be guaranteed.

6. Conclusions and Future Work

Even though real-time synchrony protocols significantly reduce the design and verification complexity of distributed real-time systems, they have not been widely applied to real systems due to their limiting conditions. To address this problem, in this paper, we proposed the *Quality-Aware PALS (qPALS)* protocol that makes synchronous computing models more feasible in a loosely coupled distributed systems without a fault-tolerant real-time network. qPALS supports flexible synchronization semantics, allowing each synchrony class serving different purposes: performance and correctness of synchronization. Further, they can be composed in a single synchronization group to achieve both performance and strict consistency. Through the simulation, we showed that qPALS can maintain these benefits even when the underlying network environment is not stable.

qPALS is being implemented as an extension of PRISM middleware [11] that is the first prototype implementation of PALS protocol. We plan to apply the extended PRISM middleware to other CPS, such as Medical Device Plug-and-Play [7], in which distributed consistency is required for safety, but no fault-tolerant real-time network is available.

References

[1] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, Jan 2003. Article (CrossRef Link)

[2] S. Miller, D. Cofer, L. Sha, J. Meseguer, and A. Al-Nayeem, "Implementing logical synchrony in integrated modular avionics," in *Proc. of Digital Avionics Systems Conference*, 2009. DASC '09. IEEE/AIAA 28th, pp. 1.A.3–1 –1.A.3–12, oct.2009. Article (CrossRef Link)

[3] N. Leveson and C. Turner, "An investigation of the therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18 –41, July 1993. Article (CrossRef Link)

[4] G. Berry and G. Gonthier, "The esterel synchronous programming language: design, semantics, implementation," *Sci. Comput. Program.*, vol. 19, no. 2, pp. 87–152, Nov. 1992. Article (CrossRef Link)

[5] S. Andalam, P.S. Roop, A. Girault, "Deterministic, predictable and light-weight multithreading using PRET-C," *Design, Automation & Test in Europe Conference & Exhibition* (DATE), 2010, pp. 1653-1656, IEEE, 2010 Article (CrossRef Link)

[6] L. Sha, A. Al-Nayeem, M. Sun, J. Meseguer, and P. C. Olveczky, "PALS: Physically Asynchronous Logically Synchronous Systems," University of Illinois at Urbana-Champaign, http://www.ideals.illinois.edu/handle/2142/11897, Tech. Rep., 2009.

[7] "MD PnP Project: Getting connected for patient safety, http://www.mdpnp.org," 2012.

[8] C. Kim, M. Sun, S. Mohan, H. Yun, L. Sha, and T. F. Abdelzaher, "A framework for the safe interoperability of medical devices in the presence of network failures," in *Proc. of Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '10. ACM, pp. 149–158, 2010. Article (CrossRef Link)

[9] J. Meseguer and P. C. ¨Olveczky, "Formalization and correctness of the PALS architectural pattern for distributed real-time systems," in *Proc. of Proceedings of the 12th international conference on Formal engineering methods and software engineering*, ICFEM'10. Berlin, Heidelberg: Springer-Verlag, pp. 303–320, 2010. Article (CrossRef Link)

[10] S. Ranganathan, A. D. George, R. W. Todd, and M. C. Chidester, "Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters," *Cluster Computing*, vol. 4, no. 3, pp. 197–209, Jul. 2001. Article (CrossRef Link)

[11] A. Al-Nayeem, C. Kim, W. Kang, P.-L. Wu, and L. Sha, "Middleware design for physically-asynchronous logically-synchronous (pals) systems," in *Proc. of Proceedings of the 13th ACM international conference on Embedded software (Emsoft'13)*, 2013. Article (CrossRef Link)

[12] N. Kottenstette, X. Koutsoukos, J. Hall, J. Sztipanovits, and P. Antsaklis, "Passivity-based design of wireless networked control systems for robustness to time-varying delays," in *Proc. of Proceedings of the 2008 Real-Time Systems Symposium*, RTSS '08, 2008. Article (CrossRef Link)

[13] W. Kang, K. Kapitanova, and S. H. Son, "RDDS: A real-time data distribution service for cyber-physical systems," *IEEE Trans. Industrial Informatics*, vol. 8, no. 2, pp. 393–405, 2012. Article (CrossRef Link)

[14] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive stream resource management using kalman filters," in *Proc. of* SIGMOD '04: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, pp. 11–22, 2004. Article (CrossRef Link)

[15] V. Fay-Wolfe, L. C. DiPippo, G. Cooper, R. Johnson, P. Kortmann, and B. Thuraisingham. "Real-time CORBA," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 11, no. 10, pp. 1073-1089, 2000. Article (CrossRef Link)

[16] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Trans. Comput. Syst*., vol. 17, no. 2, pp. 41–88, May 1999. Article (CrossRef Link)
[17] A. Al-Nayeem, L. Sha, D. D. Cofer, and S. P. Miller, "Pattern-based composition and analysis of virtually synchronized real-time distributed systems," in *Proc. of Proceedings of 3rd International Conference on Cyber-Physical Systems (ICCPS)*, 2012. Article (CrossRef Link)
[18] R. Obermaisser, "Reuse of can-based legacy applications in time triggered architectures," *IEEE Transactions on Industrial Informatics*, vol. 2, no. 4, pp. 255–268, 2006. Article (CrossRef Link)
[19] W. Steiner and J. Rushby, "TTA and PALS: Formally verified design patterns for distributed cyber-physical systems," in *Proc. of Digital Avionics Systems Conference (DASC)*, 2011 IEEE/AIAA 30th, oct. 2011, pp. 79-86, 2011. Article (CrossRef Link)
[20] K. P. Birman, "Replication and fault-tolerance in the ISIS system," *ACM SIGOPS Operating Systems Review*, vol. 19, no. 5, pp. 79–86, 1985. Article (CrossRef Link)
[21] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64-83. 2003. Article (CrossRef Link)
[22] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight probabilistic broadcast," *ACM Trans. Comput. Syst*., vol. 21, no. 4, pp. 341–374, Nov. 2003. Article (CrossRef Link)
[23] A. Kermarrec and M. Steen. "Gossiping in distributed systems," *SIGOPS Oper. Syst. Rev.* vol. 41, no. 5, pp. 2-7, October 2007. Article (CrossRef Link)
[24] J. Pereira, R. Luís, and O. Rui. "Semantically reliable multicast: Definition, implementation, and performance evaluation," *Computers, IEEE Transactions on*, vol. 52, no.2, pp. 150-165, 2003. Article (CrossRef Link)

Woochul Kang received his PhD degree in computer science from the University of Virginia in 2009. He joined Incheon National University as an assistant professor in 2013. His research interests include cyber-physical systems, real-time embedded systems, large-scale distributed systems, and feedback control of computing systems. He was a senior researcher at Electronics and Telecommunications Research Institute (South Korea, 2000-2004, 2009-2012), and post-doc research associate at the University of Illinois at Urbana-Champaign (USA, 2012-2013).

Lui Sha received the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, in 1985. He is currently a Donald B. Gillies Chair Professor of computer science at the University of Illinois at Urbana Champaign. His work on real-time computing is supported by most of the open standards in real-time computing and has been cited as a key element to the success ofmany national high-technology projects including GPS upgrade, the Mars Pathfinder, and the International Space Station. Dr. Sha is a Fellow of the Association for Computing Machinery (ACM).