Sustaining Property Verification of Synchronous Dependable Protocols Over Implementation*

Péter Bokor[1,2], Marco Serafini[1], Áron Sisak[2], András Pataricza[2] and Neeraj Suri[1]

Technical University of Darmstadt, Germany[1]
{pbokor,marco,suri}@informatik.tu-darmstadt.de

Technical University of Budapest, Hungary[2]
{petbokor,sisak,pataric}@mit.bme.hu

Abstract

It is often considered that a protocol that has been verified for its dependability properties at the protocol level maintains these proven properties over its implementation. Focusing on synchronous protocols, we demonstrate that this assumption can easily be fallacious. We utilize the example of an existing formally verified diagnostic protocol as implemented onto the targeted time-triggered architecture (TTA). The cause is identified as the overlap mismatch across the computation and communication phases in TTA, which does not match the system assumptions of the protocol. To address this mismatch problem, we develop the concept of a generic alignment (co-ordination) layer to implement the desired communication assumptions. The verification of this layer ensures that the formally proved properties of a protocol also hold over their varied implementation.

1. Introduction

To manage the complexity of formal verification of distributed protocols, a divide-and-conquer approach is often advocated. For synchronous round-based protocols implemented on the targeted time-triggered (TT) systems, a general approach is to first verify the protocol based on its abstract specification, and then to prove that, under specified design constraints, the TT implementation of the protocol preserves the verified properties. By first establishing that the platform implements the system assumptions made by the protocol, it is possible to prove a *simulation relation* between the abstract protocol definition and the model of the TT implementation validating that the two models proceed through the same sequence of system states [12]. The first step must be repeated for each single protocol, whereas the second can be done once for all, significantly reducing verification efforts.

*Research supported in part by EC DECOS & ReSIST

In this paper we elaborate how the Time-Triggered Architecture (TTA) [6] is able to support the execution of a synchronous round-based protocol preserving the protocol's fault-tolerance properties. *Time-triggered systems* can be broadly defined as systems where all nodes have a synchronized time basis and they execute desired operations at specific times following an a-priori deterministic schedule. We contrast TTA with other time-triggered systems that assume more co-ordination among the nodes, thus facilitating a property preserving implementation of synchronous round-based protocols.

In TTA, the nodes access a *shared* communication medium using a Time Division Multiple Access (TDMA) scheme to determine the time when a node is allowed to send messages on the bus. This is in contrast with *frame-based* TT systems (e.g., [5]), where nodes are assumed to send messages via dedicated channels at approximately the same time following a sequence of communication and computation phases. The latter behavior directly implements the main assumption of synchronous round-based algorithms[1]. In TTA, in order to minimize the latency of control applications, the computation phase of individual nodes is staggered over the time of communication of other nodes in the cluster. As a consequence, the constraints defined for property preservation of frame-based TT implementation of synchronous round-based protocols [12] are *not* met by TTA.

Contributions. To substantiate this claim, we consider a formally verified, synchronous round-based diagnostic protocol and show that, in its original form, it fails to guarantee the desired behavior in a TTA system. We use model checking to obtain counterexamples violating the correctness of the algorithm.

We then introduce a generic *alignment layer* concept that can be used to simulate the communication properties assumed by the protocol. The example of the diagnosis al-

[1]We use "protocol" and "algorithm" as synonyms, following the different terminology in the referred literature. Similarly, in some places, dependability and fault-tolerance are used interchangeably.

gorithm and the TTA platform are used as a case study of the general concept. In particular, it is shown how using the alignment layer, the original round-based algorithm can satisfy the desired fault-tolerance properties in the TTA model.

Overall, a framework is proposed to ease the verification of abstract programs over their implementation. As a general example, we prove that the alignment layer on top of TTA yields the assumptions of synchronous, round-based protocols, thus sustaining their original properties. The general framework is not restricted to a specific class of protocols, neither to the alignment layer (which can provide various co-ordination), nor to the underlying implementation platform. In fact, the time-triggered implementation of round-based protocols over frame-based systems [12] and over the TTA (as presented) can be treated as special cases.

If the alignment layer can be verified, the framework not only ensures that the existing formal proof of a protocol automatically transforms to the protocol's implementation, but also facilitates the design of new fault-tolerant algorithms. During the formal analysis of the algorithm, the implementation specific details need not be reflected in the formal model, since the alignment layer guarantees that the protocol's assumptions are met. For example, instead of explicitly modeling the TTA behavior (as in [1, 11]), an abstract model can be defined based on the protocol's assumptions, which yields (i) a reduced size of the state space (mitigating state space explosion in automated verification), and (ii) a simpler and less error-prone design.

Organization. In Section 2, the frame-based and TTA models of time-triggered systems are contrasted for their implementation of round-based algorithms. In Section 3, we perform formal analysis of a round-based diagnostic algorithm to elaborate its correctness in both time-triggered implementations. We also present the fault model and verify that the TTA implementation of the algorithm violates the specified requirements. A message alignment layer is proposed as a solution. Section 4 presents a framework to formally verified implementation of abstract programs. We prove that message alignment is able to provide the assumptions of round-based protocols, and validate the soundness of the proposed framework via our case study. Section 5 discusses the benefits of the approach based on experimental data. In Section 6, the results are compared with related work, and Section 7 concludes the paper.

2. Flavors of Time-Triggered Systems: Implementing Round-Based Protocols

In [8], Lynch identifies a class of distributed algorithms, where the nodes of the algorithm proceed through rounds, thus, we call them *round-based* algorithms. Within a round every node executes the following two phases: In the *communication phase*, messages are sent to the other nodes,

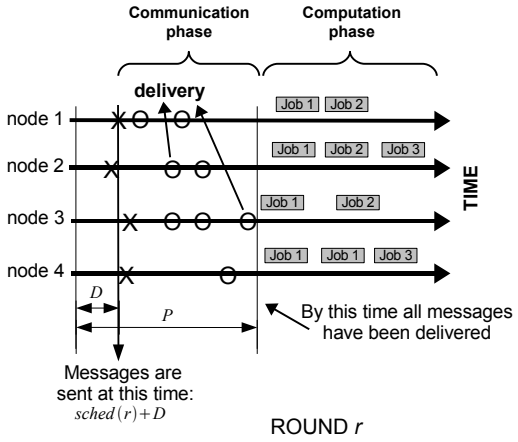

Figure 1. The frame-based TT model: Communication and computation alternate

which deliver these messages; messages are computed by the local message generation function. Next, in the *computation phase*, the local state of a node is updated by the state transition function based on the messages received from the other nodes. As it is assumed that nodes execute the different phases in lock-step, we call such systems *synchronous*. Systems assuming that concurrent actions execute at exactly the same instant of real time (perfect synchrony) are also termed *untimed* (as in [12]) reflecting that the notion of time can be abstracted and, as such, is not needed.

However, synchronous systems in real applications can achieve only a certain extent of symmetry, and the untimed abstraction must be thus implemented. In the following sections we discuss how this abstraction is obtained in two different classes of TT systems using different communication and computational paradigms.

2.1. Frame-Based Time-Triggered Systems

The *frame-based* class of TT systems (e.g., [5]) considered by Rushby in [12] is very close to the synchronous round-based model of computation (Figure 1). Nodes proceed through rounds concurrently, and communication and computation blocks can be globally separated in each such round. Every node is allowed to send messages to the other nodes, and every message from correct processes is assumed to be delivered within bounded time. Computation begins after all nodes have sent the messages and all messages have been delivered. Every node then executes local *jobs*, which generate the messages to be sent in the next round based on the collection of messages currently available in the input channels. A job can be any application running (distributed) on the nodes (e.g., implementing clock synchronization).

As TT systems are synchronous, there are known upper bounds on the execution time of jobs and on communica-

tion delays among correct nodes. Each node is equipped with a local clock of known precision, i.e., the *drift* between clock time and physical time is bounded. Although a clock synchronization protocol is used to keep the local clock of nodes close to each other, it cannot be guaranteed that they show exactly the same time. Clock synchronization guarantees that the values of local clocks periodically converge and the difference of local clock times (called the *skew*) is bounded. As a consequence, different nodes start a synchronized action at slightly different physical times (e.g., varying times of sending the messages in Figure 1).

The motivation of Rushby's work [12] is to abstract the details of (time-triggered) implementation of round-based protocols, such as synchronization of local clocks, whilst conducting formal analysis to verify correctness and fault-tolerance. Rushby concludes that the formal analysis might fail to identify the basic mechanisms of *why* the algorithm is fault-tolerant, if the formal model of the algorithm contains implementation-related details (like in [7], for example). In order to establish a systematic framework, a *simulation relation* is defined between the execution of a round-based algorithm and its frame-based TT implementation, i.e., it is proven that the implementation proceeds through the same sequence of "global states" (the array of local states of the nodes at each round) as the round-based algorithm.

According to Rushby's model, each node initiates round r at local time $sched(r)$ and begins the communication and computation phases after global fixed offsets respectively called D and P (see Figure 1). To reproduce the lock-step behavior assumed by synchronous round-based protocols, these offsets must be large enough to allow each node the time needed to complete one phase before any other node initiates the next. The paper identifies necessary constraints (on the offsets D and P, clock drift and skew, etc.) to guarantee the simulation relation. We show that this model, while in accordance with frame-based TT systems, does not match the communication and computation model of the TTA.

2.2. Time-Triggered Architecture (TTA)

The TTA is also a time-triggered system but has different communication and computation patterns than frame-based systems. The scheduling of each node in the TTA model is not required to follow a global parallel sequence of communication and computation phases (Figure 2). The TDMA access scheme assigns a different *sending slot* to each node periodically, once in every TDMA round. This is called *global schedule* with the convention that the i^{th} sending slot is assigned to node i. As the system is synchronous, the messages sent by node i will be delivered by all other

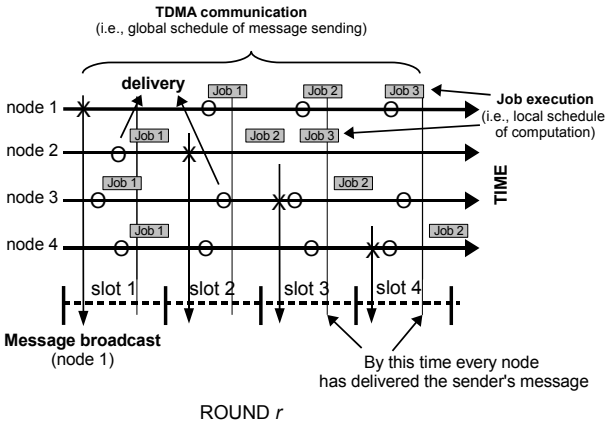

Figure 2. The TTA model: Communication and computation overlap

nodes within a bounded time[2]. Time division multiplexing is needed, because TTA systems use a shared (duplicated) bus. It is also assumed that messages are *broadcast* to all nodes.

Besides the global communication scheduling, each node has an internal *node schedule* determining when jobs are executed. This schedule is independent of the TDMA scheme, and it is computed to achieve optimal execution using different measures (e.g., minimize error propagation or execution time of a distributed application).

Note that the frame-based model do not apply to TTA systems, as $sched(r)$, D and P are not globally defined. The communication phase of different nodes is staggered over a TDMA round overlapped with the execution of jobs (which corresponds to the computation phase). Consequently, implementations of a synchronous round-based protocols on TTA systems do not (in general) guarantee the desired properties, as shown in the following section. In [13] we present *message alignment* to remedy the overlap of different phases in case of a round-based diagnostic and membership protocol. We now show that the same mechanism can be used in TTA for the correct implementation of any round-based algorithm. The intuition is to introduce an additional latency in processing the messages, based on the fact that all messages sent in round $r-1$ are available in every node at round r. In the following section, the approach is illustrated by an example protocol, and the results are validated via model checking.

[2]Due to the drift and skew of the local clocks, and the latency of message delivery, the global schedule is determined such that correct nodes agree on the current slot.

3. Case Study: Verifying Inconsistency of a Round-Based Diagnostic Protocol on TTA

In [16] the authors define and formally verify the synchronous round-based algorithm Distributed Diagnosis (DD)[3]. As discussed earlier, the fact that DD can be formally verified implies that its frame-based implementation also guarantees the verified properties [12]. We will use DD to substantiate that the implementation of an abstract protocol on the TTA might not exhibit the desired properties.

Algorithm DD executes in two rounds, each split into communication and computation phases:

Round 1 - Local Detection

1. *Sending workload (communication):* Every node broadcast its workload message(s) generated by the job(s) running on the node.
2. *Local detection (computation):* Every node (locally) diagnoses the other nodes based on the incoming messages and forms a *local syndrome* indicating the faulty/correct status of each node.

Round 2 - Dissemination and Analysis

1. *Global dispersion (communication):* Every node broadcasts its local syndrome.
2. *Global assimilation (computation):* The local syndromes received from the other nodes, together with the syndrome obtained in Round 1, are compiled into a *syndrome matrix*. The i^{th} row of the matrix is the local syndrome received from the i^{th} node.
3. *Analysis (computation):* Every node computes the majority among the values of each column of the syndrome matrix[4] to derive the *global health vector*. Each value of this vector indicates the faulty/correct status (binary value) of the relative node.

In the following, the fault model and the desired properties of DD will be defined.

3.1. Fault-Tolerant Distributed Diagnosis

The properties of fault-tolerant protocols need to be proven against a fault model. In a *hybrid* fault model, pessimistic fault modes are considered together with less severe fault manifestations. As reported in many applications (e.g., [13]), the classification of hybrid fault classes contributes to a better fault coverage and a finer diagnostic resolution. DD assumes a hybrid fault model, where no assumption is made about *malicious* (or Byzantine) nodes (i.e., they can send arbitrary messages), and *benign* nodes are always detected by every correct node based on the erroneous/missing messages sent by the faulty node (e.g., crash faults). If the maximum number of malicious and benign nodes are respectively denoted by a and b, the fault hypothesis (i.e., the number of tolerated faults) of DD defines $N > 2a + b + 1$, where $a \leq 1$ and N is the overall number of nodes. For more details about the fault model of DD, the reader is referred to [16].

The fault model of DD can be directly justified in the TTA architecture. Symmetric faults are a consequence of the bus topology, however, asymmetric faults (e.g., [1],[13]) and even Byzantine behavior [4] are also possible in TTA implementations.

Diagnostic protocols usually require *correctness* (i.e., correct nodes are never diagnosed as faulty), *completeness* (i.e., faults are eventually detected) and *consistency*. To prove that DD (in its original form) is not correct under the TTA, we show that consistency is violated.

Consistency All correct nodes agree on the global health vector.

We remark that correctness and completeness are also violated, however, for the sake of our argumentation we restrict ourselves to consistency.

3.2. Implementation Problems on the TTA

We show that DD, although being formally verified under the untimed model [16], violates consistency in the TTA. We use the SAL symbolic model checker [3] to analyze DD and its modifications in the TTA. First a counterexample is shown as a run in DD, such that a node is diagnosed differently by two correct nodes (thus violating consistency).

In the SAL model of DD in the TTA, we assume that the diagnostic job is executed once in every TDMA round. Accordingly, all nodes have the chance to send messages between two executions of a diagnostic job. These messages are then used to set up the new diagnostic view. The following abstraction is used: If the execution of a node's diagnostic job is scheduled after slot j, the messages sent in the current round are available from nodes $[1, k]$, where $k \leq j$. The parameter k depends on how messages are delivered from the communication layer to the application (e.g., considering the delay caused by error detection mechanisms). Similarly, the new message (as the output of the diagnostic job) is available at the communication controller from slot l on, where $l \geq (j + 1)$. The parameter l is primarily determined by the execution time of the diagnostic

[3] DD is chosen for its representativeness as a synchronous protocol, and also given the public availability of its detailed specification and formal verification.

[4] In fact, *hybrid majority* is computed, i.e., incorrectly delivered local syndromes and the node's opinion on itself are omitted in the voting; the default is "faulty" if no value is in majority.

job. In particular, we considered the following special case, where `dj[i]` indicates the scheduling of the diagnostic job at node *i*. If `dj[i]=j` ($j \in [0, N-1]$), the diagnostic job of node i can read the messages from nodes $[1, j]$ and the new message is available to be sent from slot $(j+1)$ on. `dj[i]=0` means that the diagnostic job is executed after the last sending slot of a TDMA round, and it can read data from all nodes in the last round and send new messages in the following round.

In the following examples we show that different node schedules of the diagnostics jobs can result in inconsistencies of the global health vector. Once again, the node schedule of all applications running on the TTA cluster is calculated and optimized a-priori by a scheduler. Diagnosis is possibly a low priority application, no assumption can thus be made about the scheduling of diagnostic jobs. For simplicity, we assume (**Assumption 1**) that the diagnostic job is executed *after* the sending slot of the node within each TDMA round (e.g., Job 2 in Figure 2).

The SAL model is called `DD`, and `consistency` is formulated as an invariant using the temporal logic operator G. The following theorem requires that the global health vector of any two correct nodes (`ghv[i]` and `ghv[j]`) are always consistent:

```
consistency: THEOREM
  DD |- G(FORALL(i,j,k:node):
    (non_faulty(i) AND non_faulty(j) =>
       ghv[i][k] = ghv[j][k]));
```

SAL fails to prove the theorem and produces the following counterexample ($N = 4$). It can be seen that the node schedule fulfills Assumption 1, since `dj[i]` $\leq$ `i`, for all i. The SAL variable `fv[i]` indicates the fault status of node i. For simplicity, it is assumed that faults are permanent. In the counterexample, node 2 is benign faulty at round $(r-1)$, otherwise all nodes are correct. Consistency is violated, because node 1 disagrees with node 4 on the diagnosis of node 2.

```
Counterexample 1:
=========================
...
%Node schedule of diagnostic jobs
dj[1] = 1
dj[2] = 2
dj[3] = 3
dj[4] = 0
...
%Fault vector in round r-1
fv[1] = non_faulty
fv[2] = benign
fv[3] = non_faulty
fv[4] = non_faulty
...
%Result of diagnosis at round r
ghv[1][1] = non_faulty
ghv[1][2] = non_faulty %INCONSISTENCY
ghv[1][3] = non_faulty
ghv[1][4] = non_faulty
...
ghv[4][1] = non_faulty
ghv[4][2] = faulty    %INCONSISTENCY
ghv[4][3] = non_faulty
ghv[4][4] = non_faulty
```

Inconsistency is caused by the overlap of communication and computation in TTA. As the diagnostic jobs running on nodes 1 and 4 are scheduled respectively before and after the sending slot of the faulty node (2), 1 detects the fault later than 4. The run leading to inconsistent diagnosis at round r is the following. Node 1 reads local syndromes sent in round $r-1$. None of nodes 3 and 4 can report the fault in round $r-1$, thus, they build a majority ($N = 4$). Therefore, node 2 is deemed as non-faulty by node 1. On the other hand, node 4 detects node 2 as faulty at round r and receives another accusation from node 3. This is sufficient to deem 2 as faulty. Note that node 2's self-syndrome is omitted in the majority voting.

Read alignment We now introduce an alignment layer as a remedy to the problem identified by Counterexample 1. The reason why DD misbehaves in the TTA model is that the diagnostic information of two successive rounds overlap. In fact, different nodes use diagnostic information related to different rounds depending on the time when diagnostic jobs are executed. This is possible due to the unconstrained node schedule.

A mechanism called *read alignment* is used to delineate diagnosis of different rounds [13]. The main idea is that every node buffers the messages sent by other nodes, and computation is done based on messages that can be consistently read by all jobs. Assume that a diagnostic job reads the diagnostic messages $\{m_1, ..., m_N\}$ from all nodes, and the node schedule is defined by l_i with the same semantics as `dj[i]`. It is assumed that messages m_i are updated in the order of sending, following the TDMA scheme. To let all diagnostic jobs use consistent diagnostic information (i.e., data of the same freshness), a read alignment layer is assumed providing the following service: Messages $\{m_1, ..., m_{l_i}\}$ as read in the previous round, and $\{m_{l_i+1}, ..., m_N\}$ as read in the current round, are collected by the alignment layer and sent to the node. Every time a diagnostic job is executed, it invokes the message alignment layer to access consistent diagnostic data, instead of reading messages directly via the TDMA communication.

To rectify inconsistency of DD in the TTA, we install the read alignment layer between the TTA communication layer and the application layer (see Figure 3). Accordingly, diagnostic jobs access data from the read alignment layer in both rounds of the protocol: During local detection, every node detects faults of other nodes based on incoming messages from the previous TDMA round, and local syndromes

are computed accordingly. During global assimilation, local syndromes sent in the previous round are collected in the syndrome matrix, which is used to calculate majority.

Further problems We augmented DD with the model of the read alignment layer, removed Assumption 1 and ran model checking again. As a result, consistency still cannot be proven and the following counterexample is provided.

```
Counterexample 2:
=========================
...
%Node schedule of diagnostic jobs
dj[1] = 1
dj[2] = 2
dj[3] = 3
dj[4] = 4
dj[5] = 4
...
%Fault vector in round r-2
fv[1] = non_faulty
fv[2] = benign
fv[3] = non_faulty
fv[4] = non_faulty
fv[5] = non_faulty
...
%Fault vector in round r-1
fv[1] = malicious
fv[2] = benign
fv[3] = non_faulty
fv[4] = non_faulty
fv[5] = non_faulty
...
%Result of diagnosis at round r
...
ghv[4][1] = non_faulty
ghv[4][2] = non_faulty %INCONSISTENCY
ghv[4][3] = non_faulty
ghv[4][4] = non_faulty
ghv[4][5] = non_faulty
...
ghv[5][1] = non_faulty
ghv[5][2] = faulty     %INCONSISTENCY
ghv[5][3] = non_faulty
ghv[5][4] = non_faulty
ghv[5][5] = non_faulty
```

Prior to examining the given execution of the protocol, observe that the number of nodes has been increased to 5 to tolerate one benign and one malicious node at the same time. Note that Assumption 1 is violated, as node 5 always executes the diagnostic job before its sending slot (dj[5]=4), which means that fresh diagnostic information is sent. Consistency is violated as node 4 and 5 disagree on the diagnosis of node 2 in round r.

The counterexample is the following: At round $(r-1)$, local detection is done based on read aligned messages from round $(r-2)$. As node 2 is faulty in that round, all correct nodes build an updated local syndrome containing the accusation on node 2 (denote it by $faulty(2)$). According to the node schedule, nodes 3 and 4 cannot immediately send the updated local syndromes, thus, they send the prior local syndromes referring to round $(r-3)$ containing $correct(2)$. On the other hand, node 5 can immediately send the updated local syndrome with $faulty(2)$. Furthermore, the malicious node 1 can send local syndromes containing $correct(2)$ and $faulty(2)$ to node 4 and 5, respectively.

At round r, the local syndromes sent in round $(r-1)$ are aggregated (read alignment) and used to calculate the majority. Accordingly, node 4 collects the values $\langle correct(2), x, correct(2), correct(2), faulty(2)\rangle$ from nodes 1 to 5 respectively (node 2's opinion about itself is denoted by 'x', since it is not considered in the voting). The diagnosis of node 2 obtained by node 4 through majority voting is thus $correct(2)$. On the other hand, node 5 receives $\langle faulty(2), x, correct(2), correct(2), faulty(2)\rangle$ and since no value is in majority, the default value $faulty(2)$ is adopted to diagnose node 2.

Send alignment In the example above, nodes 3 and 4 broadcast local diagnosis about node 2 referring to round $(r-3)$, whereas node 5 sends data from round $(r-2)$, which leads to inconsistency. Therefore, *send alignment* can be used to delay the sending of fresh data if needed [13]. In fact, if all nodes can send fresh data or, equivalently, none of them can send fresh data (like in Assumption 1), such a mechanism is not needed, since all data refer to the same TDMA round. On the other hand, if $l_i \geq i$ (i.e., node i cannot send fresh data) and $l_j < j$ (i.e., node j can send fresh data), for some i and j, send alignment is needed to guarantee consistency: At every round, node i forms and immediately sends the local syndrome based on the messages received from the read alignment layer, however, node j sends the syndrome based on the messages received in the previous TDMA round, thus, delaying the broadcast of the fresh local syndrome.

Accordingly, local syndromes assimilated and analyzed at round r were all sent at the previous round (read alignment), and they all refer to round $(r-2)$ or to round $(r-3)$ (send alignment). The layer incorporating both read and send alignment is called the *message alignment layer*. Note that the minimum delay imposed by the message alignment layer is two TDMA rounds, which can be guaranteed if $l_i < i$, for all i, and send alignment is not needed. In the following section, we prove that any round-based algorithm maintains its original requirements, if it is deployed on the TTA platform augmented with the message alignment layer.

4. The Alignment Layer: Transformation to Provide Equivalence

In this section, we propose a framework to formally verify abstract programs over their implementation (Sec-

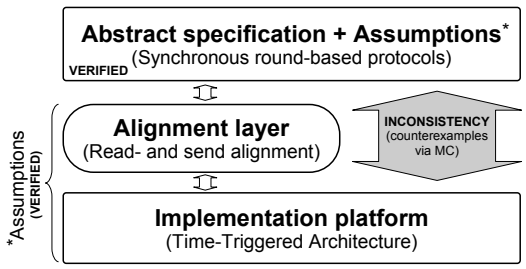

Figure 3. Verification of abstract programs over their implementation

tion 4.1). As a general example, we show how the framework applies to round-based, synchronous protocols implemented over the TTA (Section 4.2). Finally, as a validation of the overall approach, the results are demonstrated on the example case study (Section 4.3).

4.1. A Framework To Verify Abstract Programs over Implementation

Figure 3 depicts the main elements of the proposed verification framework. Given the *abstract specification* of a protocol together with the system's *assumptions* (communication, synchrony, etc.), the protocol is to be implemented on a specific *platform* such that the original specification is not violated. The implementation platform possibly deviates from the protocol's assumption. In such cases an *alignment layer* is necessary. We propose to formally prove that the layer on top of the given platform satisfies the assumptions. This is in contrast to formally verifying the *composite model* comprising the models of the protocol (with the fault model) and the implementation platform. It is assumed that the abstract specification is formally *verified*, thus, it is free from design faults. Accordingly, the implementation of new protocols on the same platform can be done without verifying the alignment layer again. In Figure 3, our prototype verification settings are written in brackets, i.e., model checking synchronous, round-based protocols over the TTA. The next section proves that read and send alignment in TTA indeed guarantee the desired assumptions.

We remark that the implementation of round-based protocols on a frame-based, time-triggered platform similarly conforms to the proposed framework. In this case, the alignment layer identifies necessary constraints posed on the frame-based system, e.g., constraints on the skew and drift of local clocks, which in turn relate to the applied clock synchronization algorithm. The lock-step behavior of untimed systems can be provably simulated, if these constraints hold [12]. Finally, note that formal techniques applied by the framework are not specified. In particular, we used model checking to verify the abstract specification of the protocol. In [12] theorem proving is utilized for the same purpose. In both cases, the properties of the alignment layer are proven manually.

4.2. A Proof: Message Alignment over TTA Guarantees the Lock-Step Assumption

In this section, we prove that the message alignment layer ensures that the assumption of parallel alternation of communication and computation phases at different nodes is held on top of the TTA. Therefore, any round-based algorithm sustains its properties if implemented over the TTA. We use induction to show that the TTA implementation proceeds through the same sequence of states as the round-based protocol, however, with a delay of one or two TDMA rounds. This means that the TTA-implementation of a round-based protocol with r rounds terminates after $2(r-1)$ (without send alignment) or $3(r-1)$ TDMA rounds (with send alignment), depending on the node schedule[5]. The system's state is naturally defined as the array of local states of every node (similarly to [12]).

We now formally define synchronous, round-based protocols termed as *untimed systems* after [12]. This mathematical description will be the base of the equivalence proof. Note that TTA systems use broadcasting, hence, we restrict to broadcast communication, as opposed to the general class of synchronous, round-based protocols [8].

Definition 1. (Untimed system). *An untimed system (*UT system *in short) consists of a set of* nodes *P (denote N the number of nodes) maintaining the set of* states *S, and a set of* messages *M. Node i defines a* state transition function[6] $t_i : S \times M^N \rightarrow S$*, and a* message generation function $m_i : S \rightarrow M$*. A set of* initial states *is defined by* $S^I \subseteq S$*.*

Every node starts from an initial state. The system makes the following two phases in lock-step, repeatedly in round *1,2, etc.: In the* communication phase *each node i generates a message (via* m_i*) that are sent to the other nodes (broadcast communication); in the* computation phase *node i updates its next state (via* t_i*) based on the messages received in the current round. The* global state *of the system at round* r *is the array of each node's state, and it is denoted by* $ut(r)$*.*

We now formally define the TTA-based implementation of an untimed system. This means that the same functions defining the protocol's operational behavior are assumed. However, the communication follows the TDMA scheme. The alignment layer implementing read and send alignment on top of the TTA is also included in the definition. Messages are received and sent via this alignment layer. In the

[5] In the initial round there is no receipt of messages, and similarly, no messages are sent in the final round. Therefore, the overall number of TDMA rounds is less than $2r$ and $3r$.

[6] For simplicity, we assume that a node receives its own message.

mathematical description, we use a similar model to the one introduced in [13]. A communication controller attached to each node is responsible for sending and receiving the messages. The communication controller maintains *interface variables* $\{m_1, ..., m_N\}$, where m_i is the message last read from node i. Interface variables are updated at every sending slot i with the message sent by node i. Note that the following model is also "untimed" in the sense that no notion of (local) clocks appear in the definition (as opposed to the definition of time-triggered systems in [12]). Although, a TTA system is implemented based on synchronized local clocks, these details are abstracted in our model, since we assume that the TTA implementation is correct.

Definition 2. (TTA system). *In the TTA system, P, S, S^I, M, t_i and m_i are defined as those in the UT system. The* node schedule *is defined by $l_i \in [0, N-1]$ for each node i.*

Every node i starts from an initial state and executes the following operations, repeatedly in TDMA round *1,2,etc. Send $m_i(s_{i,r})$ to the communication controller, if $l_i < i$, for all i ($s_{i,r}$ denotes the state of node i at round r). If $l_i \geq i$, send $m_i(s_{i,r})$, otherwise send $m_i(s_{i,r-1})$ (*send alignment*). The communication controller of node i broadcasts the message at slot i via the bus.*

The values $m_{j,r}$ ($j \in [1, N]$) are buffered storing the message received from node j at round r. The next state of node i is computed by applying read alignment*: $s_r = t_i(s_{i,r-1}, m_{1,r-1}, ..., m_{l_i,r-1}, m_{l_i+1}, ..., m_N)$. The* global state *of the system at the end of round r is the array of each node's state, and it is denoted by $tta(r)$.*

Our main result is that we prove that the UT and the TTA systems proceed through the same sequence of global states. Note that the current state of a node in the TTA system is computed based on the messages of the previous round (read alignment), thus, imposing a delay of one round. In addition, certain nodes use their previous state (rather than using the current one) to generate the message to send, if send alignment is needed. This yields an additional round of delay.

The following theorem states that the UT and the TTA system execute the same program, i.e., the abstract specification of a synchronous, round-based protocol is simulated by its time-triggered implementation. The proven equivalence also verifies that the assumptions of the protocol (such that strictly alternating sequence of communication and computation) are realized by the implementation, and the proposed framework is viable (Figure 3).

Theorem 1. *If the initial states of nodes in the UT system are the same as in the TTA system, the global state ut is simulated by the global state tta with a delay of two TDMA rounds, if there is a node i with $l_i \geq i$, and with a delay of one TDMA round otherwise:*

$$tta(r) = ut(r-1) \text{ or } tta(r) = ut(r-2)$$

Proof. *The proof is an induction on the number of rounds.* Assume that $tta(r) = ut(r-1)$. Denote $s_{i,r-1}$ and $s'_{i,r}$ the states of the i^{th} node in the UT and TTA system, at round $r-1$ and r, respectively. The messages sent in the UT system at round $r-1$ are denoted by $m_{i,r-1}$, where $m_{i,r-1} = m_i(s_{i,r-1})$. Assume that $l_i < i$, for all i, send alignment is thus not needed. Therefore, the message $m'_{i,r}$ sent by the i^{th} node in the TTA system at round r is computed by $m_i(s'_{i,r})$. By definition, the next state $s_{i,r}$ of node i in the UT system is computed by $t_i(s_{i,r-1}, m_{1,r-1}, ..., m_{N,r-1})$. The next state $s'_{i,r+1}$ of node i in the TTA system is computed by applying t_i to the messages received from the read alignment layer, $t_i(s'_{i,r}, m'_{1,r}, ..., m'_{l_i,r}, m_{l_i+1}, ..., m_N)$. From the definition of l_i, the interface variables $m_{l_i}, ..., m_N$ have not been updated with the messages sent at round $(r+1)$, thus $m_j = m'_{j,r}$. From the inductive assumption, $m_{i,r-1} = m'_{i,r}$, therefore $s_{i,r} = s'_{i,r+1}$ and the inductive proof follows.

Send alignment is needed, if $l_i \geq i$ for some i. In this case, the messages sent at round r in the TTA system are computed by $m_i(s'_{i,r-1})$. The reasoning is now similar to the previous one, and $tta(r) = ut(r-2)$ can be proven. □

4.3. A Validation of the Framework: Model Checking of DD over TTA

According to the proposed verification framework, it is not necessary to build and verify the composite model. Instead, the formal analysis of the abstract specification suffices. This approach can be justified if the model obtained by composing the models of the DD protocol, the message alignment layer, the TTA platform and the possible faults can be verified by model checking.

In fact, we created the composite model by extending `DD`. Technically, SAL's input language supports the composition of modules, where each module is a functional building block of the larger model. In order to emphasize our treatment of message alignment as a layer of the overall architecture, the new SAL model is a synchronous composition of the following sub-modules: Every diagnostic job executing DD is a module; the message alignment layer as well as TDMA communication are also modeled as different modules. The composed model is called `DD_aligned` and the SAL model checker was able to prove the following theorem:

```
consistency: THEOREM
  DD_aligned |- G(FORALL(i,j,k:node):
    (non_faulty(i) AND non_faulty(j) =>
       ghv[i][k] = ghv[j][k]));
```

N	consistency		correctness		completeness	
	UT	TTA	UT	TTA	UT	TTA
4	1s	7m	1s	3m	1s	55s
5	3s	36h 25m	2s	2h 23m	2s	1h 30m
6	10s	–	5s	13h 19m	5s	23h 57m
7	71s	–	21s	–	18s	–

Table 1. The results of model checking DD: Times of verifying the untimed system (UT) and the composite model (TTA)

Accordingly, consistency of DD can be proven in the TTA, however, as discussed in Section 3.2, only after the installation of the message alignment layer.

5. The Efforts of Verification Based on Experiments

One advantage of the proposed framework is that existing formally verified protocols provably sustain their properties over the implementation. Besides the re-usability of established solutions, the framework also supports the design and verification of new protocols. Assuming that the properties of the alignment layer are proven, the design and verification can now be performed at the abstract level. The benefit of this approach is two-fold:

(O1) The *computational complexity* of verifying the abstract specification instead of the composite model can be considerably less (even in the order of multiple magnitudes). This is due to the possibly vast number of conditional branches in the state space caused by the modeling of faults and implementation-related details.

(O2) The handling of large (and complex) models possibly yields additional complexity in the *verification* and/or *design* of the protocol.

Table 1 depicts the results of model checking consistency, correctness and completeness in the DD protocol. Experiments were done based on the model of the untimed system (UT) and the composite model (TTA) of DD, using SAL's Bounded Model Checker (BMC) installed on a 2.66 Ghz Intel processor with 2 GB memory, running Linux 2.6 kernel. No data is depicted in Table 1, if the model checker could not prove the property due to state space explosion (a timeout of 48 hours was used). It can be seen that the UT model scales even if $N \geq 7$, while the state space explodes for relative small TTA models ($N = 6$). The UT model exhibits a considerable gain, if the times of completed proofs are compared (e.g., 3 seconds versus 36 hours in case of consistency, $N = 5$). This growth of the state space in the composite model is due to the large number of combinations stemming from the possible node schedules, which have to be considered by the model checker.

Regarding (O2), Rushby already pointed out that the identification of key fault-tolerant mechanisms of distributed protocols might be tedious in a model containing implementation details [12]. In particular, the identification of such mechanisms is crucial to complete the proof, if deductive reasoning (e.g., theorem proving) is used for verification. Similarly, if the (possibly complex) formal model of the implementation platform needs to be also created, the design of the total model requires an experienced user. Finally, the verification of the composite model can affect the automation of the verification. For example, the model checking of the composite model required user interaction, if SAL's BMC model checker was used. The BMC approach is based on *k-induction* which uses a parameter (called depth) to prove a property. The depth can be usually approximated by the longest acyclic path in the model. In the UT model, the depth depends on the number of rounds, which is constant ($r = 2$). The default depth could be thus used. In case of TTA, on the other hand, the depth is also determined by the number of slots, which varies with N, and the depth of k-induction has to be tuned accordingly.

6. Related Work

Considerable work exists on finding mechanisms that guarantee safe execution of a program's implementation. A common goal is to ensure that native code provided by an untrusted party can be executed by the code consumer (e.g., an operating system kernel) without violating its safety policy. Different approaches can vary on the time of performing the safety check of the executable code, thus, affecting performance. For example, *proof-carrying code* (PCC) [10] requires the code producer to provide the executable code together with a proof of safety, which can be validated by the code consumer at deployment time (once for all). *Software Fault Isolation* [15], on the other hand, proposes instrumentation of the code and requires that safety is checked at runtime. Note that we propose a design time approach to ensure that the target architecture of a program is able to provide the specified assumptions.

Model-based development is a well established paradigm in software-engineering of embedded systems. In this context, the alignment layer can be seen as a transformation within an *automatic code generator*, whose input is a high-level, executable model of the application, and the output is the platform specific source code. In fact, we proved the correctness of this transformation containing the read and send alignment for round-based algorithms. Further validation of the code generator can be then performed using approaches like [14].

Assertion capabilities of programming languages is a general technique in software development to detect faults and provide information about their locations. As a spe-

cial case, formal specification languages (like SAL) that are usually defined to support verification at early stages of development (at design time) and at a high level of abstraction also introduce assertions. Run time approaches, on the other hand, propose monitoring assertions during execution and they often offer a simpler and more practical alternative to formal proofs of correctness [2].

The conceptual basis of *Timed Abstract Protocol Notation* (TAP) shares significant similarities with our approach [9]. TAP is a language for describing asynchronous, message passing protocols, whose semantics is defined by the abstract and concrete executions models, which are proven to be equivalent. The first one is appropriate to formal reasoning about the protocol's correctness, and the latter one can directly compile into C code. The main difference to our work stems from the synchrony assumptions of the system. Asynchronous systems impose no assumptions on the message delivery (e.g., timeouts), hence, the abstract description (TAP) can be directly implemented without requiring an alignment layer.

7. Discussion and Conclusions

We analyzed the specification of a distributed diagnostic protocol with respect to different time-triggered systems as possible implementation platforms for it. Model checking was used to verify inconsistencies between the properties of the protocol's abstract specification and the properties of the implementation. We identified that the mismatch arises as the implementation cannot directly guarantee the protocol's assumptions, such as co-ordination between communication and computation.

As a solution, we have proposed creating an alignment layer able to provide the abstract assumptions on top of the selected implementation platform. The layer implements a transformation of the platform's characteristics and acts as a wrapper to convey the desired assumptions to the nodes running the distributed protocol. In particular, we have defined message alignment to compensate the communication pattern of the TTA platform, and to simulate the lock-step assumption of the diagnostic protocol.

We have shown that the previous concept can be generalized within a verification framework, containing an abstract program, the target implementation platform, and the transforming alignment layer in the middle of the architecture. In order to support an effective verification, the properties of the alignment layer can be proven for a general class of protocols. In our case study, we have proved that message alignment provides the assumptions of any synchronous, round-based protocols.

In future work, we aim at elaborating the application of a similar framework to verify the implementation of asynchronous protocols on partially synchronous systems.

References

[1] G. Bauer and M. Paulitsch. An Investigation of Membership and Clique Avoidance in TTP/C. *Proc. of SRDS*, pp. 118–124, 2000.

[2] L. A. Clarke and D. S. Rosenblum. A Historical Perspective on Runtime Assertion Checking in Software Development. *SIGSOFT Softw. Eng. Notes*, 31(3): pp. 25–37, 2006.

[3] L. de Moura *et al*. SAL 2. *Proc. of CAV*, pp. 496–500, 2004.

[4] K. Driscoll *et al*. Byzantine Fault Tolerance, from Theory to Reality. *Proc. of SAFECOMP*, pp. 235–248, 2003.

[5] R. M. Kieckhafer *et al*. The MAFT Architecture for Distributed Fault Tolerance. *IEEE Trans. Comput.*, 37(4): pp. 398–405, 1988.

[6] H. Kopetz and G. Bauer. The Time-Triggered Architecture. *Proc. of the IEEE*, 91(1): pp. 112 – 126, 2003.

[7] L. Lamport and S. Merz. Specifying and Verifying Fault-Tolerant systems. *Proc. of FTRTFT*, pp. 41–76, 1994.

[8] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[9] T. M. McGuire. *Correct Implementation of Network Protocols*. PhD thesis, Dep. of Comp. Sci., The Uni. of Texas at Austin, 2004.

[10] G. C. Necula and P. Lee. Safe Kernel Extensions without Run-time Checking. *Proc. of OSDI*, pp. 229–243, 1996.

[11] H. Pfeifer, D. Schwier, and F. W. von Henke. Formal Verification for Time-Triggered Clock Synchronization. *Proc. of DCCA—7*, pp. 207–226, 1999.

[12] J. Rushby. Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms. *IEEE Trans. Softw. Eng.*, 25(5): pp. 651–660, 1999.

[13] M. Serafini, N. Suri, J. Vinter, A. Ademaj, W. Brandstaetter, F. Tagliabó, and J. Koch. A Tunable Add-On Diagnostic Protocol for Time Triggered Systems. *Proc. of DSN*, pp. 164–174, 2007.

[14] I. Stuermer, M. Conrad, H. Doerr, and P. Pepper. Systematic Testing of Model-based Code Generators. *IEEE Trans. Softw. Eng.*, 33(9):622–634, 2007.

[15] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient Software-based Fault Isolation. In *Proc. of SOSP*, pp. 203–216, 1993.

[16] C. Walter, P. Lincoln, and N. Suri. Formally Verified On-Line Diagnosis. *IEEE Trans. Softw. Eng.*, 1997.