Formal Specification and Analysis of AFDX Redundancy Management Algorithms

Jan Täubrich[1] and Reinhard von Hanxleden[2]

[1] Philips Medical Systems DMC GmbH*** 22335, Hamburg, Röntgenstr.. 40, Germany
jan.taeubrich@philips.com
www.medical.philips.com/de/

[2] Christian-Albrechts-Universität zu Kiel, Institut für Informatik† 24098, Kiel, Olshausenstr. 40, Germany
rvh@informatik.uni-kiel.de
www.informatik.uni-kiel.de/rtsys/

Abstract. Reliable communication among avionic applications is a crucial prerequisite for today's all-electronic fly-by-wire aircraft technology. The *AFDX* switched Ethernet has been developed as a scalable, cost-effective network, based upon IEEE 802.3 Ethernet. It uses redundant links to increase the availability. Typical consensus strategies for the redundancy management task are not feasible, as they introduce too heavy delays. In this paper, we formally investigate AFDX redundancy management algorithms, making use of Lamport's Temporal Logic of Actions (TLA). Furthermore, we present our experiences made with TLA$^+$ and the TLA$^+$ model checker TLC.

Keywords. Redundancy Management, AFDX, TLA, Model Checking, Case Study

1 Introduction

Reliable communication between avionic subsystems is essential, especially as in 1988 with the Airbus A320 the *all-electronic fly-by-wire* technology attained commercial airline service. There are established avionic data communication protocols such as *ARINC 429* [2] and *MIL-STD-1553* [16]. Recently, the desire for increased performance and more cost-effective solutions has prompted the industry to also explore off-the-shelf alternatives such as IEEE 802.3 Ethernet [14]. The Ethernet specification, however, does not guarantee a maximum latency, as the package collisions are resolved through a *back off* strategy that may lead to a possibly unbounded latency. That is why the next-generation avionics data bus shall on the one hand allow usage of as much cost-efficient, IEEE 802.3 compliant hardware as possible and on the other hand shall guarantee a certain bandwidth and *Quality of Service*, which includes specifying maximal transmission latency. These requirements have lead to the ***A**vionics **F**ull **D**uplex*

*** J. Täubrich performed this work while at CAU Kiel.

† R. von Hanxleden performed part of this work while EADS Airbus, Hamburg/Toulouse

Switched Ethernet (AFDX), based upon IEEE 802.3 Ethernet technology, which is used today in the Airbus A-380.

AFDX transmits network frames over redundant networks (see Figure 1) and these redundant streams of frames get filtered at the receiving end system (Figure 2). As it turns out, the apparently simple problem of merging redundant streams of frames into a single non-redundant stream is not trivial and enforces some trade-offs in terms of availability, performance and resource requirements. A fairly thorough, but still informal investigation of this redundancy management (RM) problem has been performed in 2001 by von Hanxleden and Gambardella (documented in an internal, unpublished report [6]). This investigation appeared fairly thorough, but the apparent complexity of the problem[3] has prompted an interest in another, more formal investigation, using Lamport's Temporal Logic of Actions (TLA) [9] and the corresponding model checker TLC [17] (documented in a diploma thesis [15]).

This paper summarizes the findings of the first report and the subsequent formalization. The main contributions are, on the one hand, a formal definition and investigation of the redundancy management problem for frame-oriented communication protocols, such as AFDX, and, on the other hand, a fairly involved case study on the use of TLA and TLC for a safety-critical real-world problem.

The rest of this paper is organized as follows. The remainder of the introduction covers the basics of AFDX and TLA$^+$ and surveys related work. Section 2 presents the basics of frame ordering. Sections 3 and 4 describe an environment for the RM algorithms and the checked properties. Section 5 presents three alternative RM algorithms. Sections 6 and 7 summarize our experiences made with TLA$^+$, TLC, and the formal investigation of the RM algorithms.

1.1 AFDX

AFDX addresses the shortcomings of Ethernet using concepts of Asynchronous Transfer Mode (ATM) [5]. AFDX is a profiled network, meaning that configuration tables are loaded into switches at start-up. It is organized in a star topology with a maximum of 24 *End Systems* (ES) per switch. Larger systems can be realized through cascading.

Standard Ethernet suffers the possibility of an infinite chain of frame collisions and hence an unpredictable delay of messages. Therefore every ES is connected to a switch with two twisted pair cables, one pair for sending and the other for receiving frames, which makes AFDX full duplex. Each switch has the capability to buffer multiple packages for each ES in each communication direction. Consequently buffer-overflows and message delays due to congestion at the switch may cause erroneous behaviors. AFDX emulates a deterministic point-to-point network through the use of *Virtual Links* (VLs). Each VL builds a unidirectional path from one ES to one or maybe more other ESs. A certain predefined bandwidth is allocated for each VL, ensuring that the sum of allocations does not exceed the maximum available bandwidth of the whole network. AFDX can be run with either 10 Mbps or 100 Mbps. A minimal bandwidth and a maximum latency for end-to-end transmission is guaranteed.

[3] The original report contained 75 pages of rather terse technical writing, including 37 corollaries and 86 figures, most of which concerned with different communication scenarios. As far as such figures are meaningful, this does suggest a certain complexity of the problem.

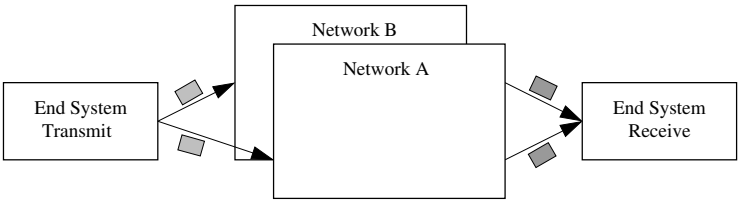

Fig. 1. Concept of redundant networks [6]

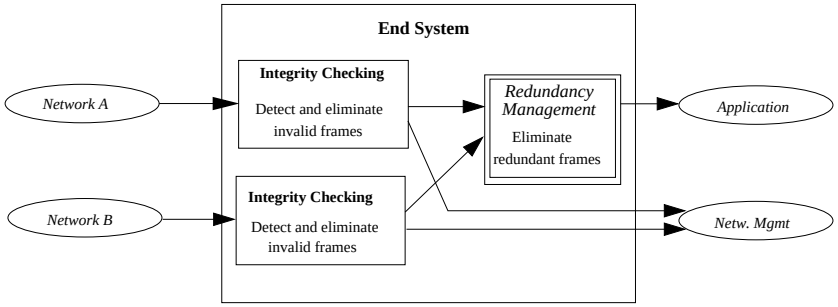

Fig. 2. Receiving End System [6]

In order to improve the reliability AFDX provides a redundant network scheme. Each frame is transmitted in parallel over two redundant networks and afterwards filtered by RM at the receiving ES (Figure 1). This shall reduce the probability of loosing frames and enable further operation even in presence of one faulty network. This redundancy has to be managed somehow, which leads to the task of *redundancy management* (RM). We consider RM to be part of the receiving ES. The task of RM can be formulated quite simply: forward all received frames to the application, but eliminate redundant copies. Hence redundancy shall be transparent to the application.

Each frame has first to pass integrity checking (see Figure 2). A basic form of integrity checking could just check if a frame is *well-formed* (e. g., whether it contains a correct CRC field). A more involved integrity check could validate whether the received frame was expected to be delivered next. Integrity checking, however, is not considered in this paper. It is assumed that integrity checking filters frames in a way that all frames reaching the RM are well-formed. No further assumptions about integrity checking are made.

1.2 Related Work

The technical problem addressed here, namely how to merge redundant streams of communicated frames (packets) into a single logical stream, should occur rather frequently in safety-critical applications. However, we have found that there is comparatively little published systematic work on this. A general approach to simplify modular specifications of dependable distributed systems is given by Sinha and Suri [12]. They propose

to define building blocks to specify and verify larger protocols. These blocks can be consensus, broadcast, redundancy management and many more. However, they do not go into further detail on how to specify these building blocks, as has been investigated in this paper for the redundancy management task.

Tanenbaum [14] gives a basic introduction about computer networks in general. Advanced topics on *ATM* and switched Ethernet are considered in Goralski [5] and Breyer [3].

Temporal logic was introduced by Pnueli [11] to describe system behaviors, a complete overview is given by Manna and Pnueli [10]. Most specifications consist of ordinary mathematics, however, temporal logic is important for describing system properties. The system properties define what a system is supposed to do and the specified automaton describes its real behavior. If the specified behavior implies the conjunction of the properties, the system behaves correctly with regard to the defined properties. In principle, a system's behavior could be defined with a single formula using this formalism. In Pnueli's logic, however, it can be hard to define certain properties of systems. TLA [8] is a variant of Pnueli's originally proposed logic. TLA was developed to permit the simplest, most direct formalization of assertional correctness proofs of concurrent systems. TLA$^+$ [9] is a specification language for concurrent and reactive systems that combines the temporal logic TLA with full first-order logic and Zermelo-Fränkel set theory. A very short introduction to the TLA$^+$ syntax is given in Lamport [7]. One main reason why we selected TLA$^+$ for formulating the redundancy management problem and associated algorithms was the ability to decompose specifications into modules and to specify reusable functions. Sommerfeld provides a case study on TLA$^+$ [13]. TLDA [1] extends TLA to compositionally specify distributed actions. TLC [17] is a model checker for debugging a TLA$^+$ specification by checking invariance and liveness properties of a finite-state model of the specification.

2 Ordering Frames

The redundancy management task requires to eliminate redundant as well as outdated frames. Consider two frames with equal content. What is needed to decide whether one frame is the redundant copy of the other? First we need to know from which *network* a frame was delivered. Two frames delivered by the same network cannot be redundant copies of each other. Furthermore an *order* of frames received from a network must be established. A common approach to identify and order frames is to include a *sequence number* (SN) in each frame. However, a problem here is that SNs are not really unique: since the number of frames sent is not a priori bounded and there are only limited resources for sequence numbers (in our case an 8-bit field), SNs eventually must wrap around. The following considerations address the problem of frame ordering with finite sequence numbers.

The *number of sequence numbers* is $SN\_CNT =_{def} 2^8$. Thus the *maximum sequence number* is $SN\_MAX =_{def} SN\_CNT - 1$. The *mid-point sequence number* is denoted as $SN\_HALF =_{def} SN\_CNT/2$. Consecutive frames have a sequence $SN(f_{i+1}) =_{def} (SN(f_i) + 1) \bmod SN\_CNT$, where SN maps frames to sequence

numbers, f_i denotes frame i, and i is some conceptual frame index denoting sending sequence.

In a first step towards comparison operators for the above defined sequence numbers we define subtraction on sequence numbers as follows:

Definition 1 *(Sequence Number Subtraction) The subtraction operator $-_{SN}$ is:*

$$s_1 -_{SN} s_2 =_{def} ((s_1 - s_2 + SN\_HALF) \bmod SN\_CNT) - SN\_HALF.$$

It can be seen that for sequence numbers within a range of SN_HALF and without a wrap around the subtraction of sequence numbers ("$-_{SN}$") is equal to common subtraction on natural numbers modulo SN_CNT and can be used to establish an order on the frames. To order sequence numbers correctly even in the presence of wrap arounds, with the known restrictions on the range of the sequence numbers, the SN_HALF gets involved into the definition. Definition 1 can be used to define the following comparison operators.

Definition 2 *(Comparison Operators)*

$$\begin{aligned} s_1 <_{SN} s_2 &\Leftrightarrow_{def} (s_1 -_{SN} s_2) < 0, \\ s_1 =_{SN} s_2 &\Leftrightarrow_{def} (s_1 -_{SN} s_2) = 0, \\ s_1 >_{SN} s_2 &\Leftrightarrow_{def} (s_1 -_{SN} s_2) > 0. \end{aligned}$$

To prove the correctness of the operators is beyond the scope of this paper, but the corollary below should enable a straight forward proof. The *unwrapped sequence number $USN(f)$* is needed to reason about sequence number operations. This number is a theoretical number for each frame f, which does not wrap around and thus is unbounded. In the following, *reset* refers to setting the sequence number count to zero; this occurs when an ES gets rebooted.

Corollary 1. *Let frames f_1 and f_2 be generated without intermediate sequence number reset, and let s_1 and s_2 be their respective sequence numbers. If $|USN(f_1) - USN(f_2)| < SN\_HALF$, then it is $s_1 -_{SN} s_2 = USN(f_1) - USN(f_2)$*

Hence a correct ordering of frames using sequence numbers can be established if the unwrapped sequence numbers differ at most by SN_HALF.

3 The Environment

In the following we describe and formally specify an appropriate environment to each of the tested redundancy management algorithms. Such an environment should feed the RM with a well-formed stream of frames and provide information to reason about the correctness of the redundancy management's decisions. The first step is to define the set of actions that the environment can perform. An appropriate environment can either send a frame, loose a frame, deliver a frame to the RM, reset the sequence number count, disable one network due to failures or it just can do nothing for a certain time.

The interaction specification of these allowed actions, hence the specified behavior of the environment without the RM part, is defined as in the TLA$^+$ fragment shown in

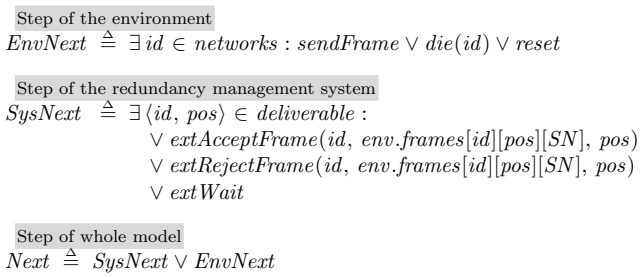

Fig. 3. Step definition of environment without RM

Figure 3. (Note: for space considerations, we here refrain from a detailed explanation of the TLA$^+$ syntax, and would like to refer the reader instead to Lamport's excellent 7-page summary of TLA$^+$ [7], which is also available on-line.) The definition of *EnvNext* expresses that a step of the environment is either a *send* step, a step where a single network *dies* (gets disabled), or a step that *resets* the sequence numbers. A frame is represented as a pair of the transmitting network id and the position in the network queue. *SysNext*, the system's next step, states that a frame from the set of deliverable frames may be either accepted or rejected, or that the environment can just do nothing. All frames ahead of the currently delivered frame on the same network are considered lost. So the loss of frames is implicitly defined, which reduces the state space significantly (and hence speeds up model checking). Finally, the definition of *Next* states that a step of the full model is either a step of the environment or a step of the system, which can be directly mapped to a step of the redundancy management.

Providing an Oracle

A crucial point of the environments specification is to enable reasoning about the correctness of the redundancy management algorithms, as the ones presented in Section 5. To recognize faulty behaviors, a system must be introduced that, independently of the actual program state, marks pending frames correctly, corresponding to their status as either *normal*, *redundant* or *old*. A frame is considered to be *normal* **iff** no frame sent later to any network has yet been received by the RM and its *twin frame* (i. e., its redundant copy) has not yet been received. A frame is considered to be *redundant* **iff** its twin frame has already been delivered to the RM. The remaining frames—which are not redundant and where a later sent frame has already been delivered to the RM—are considered *old*.

The environment sends its frames to both networks in parallel and only if both networks still have capacity to buffer another frame. Thus for a given frame f from network N_1 it is

1. decidable whether f's twin frame is still transient on N_2, and
2. possible to find the position of f's twin frame in the sequence where it is located.

Figure 4 shows how an algorithm to mark all frames correctly with a minimum effort is realized in TLA$^+$. Figure 5 shows an example behavior. The first two frames of each

$$
\begin{array}{l}
tag[seq1 \in Seq([sn : (0 \,..\, SN\_MAX),\ tag : \{\text{“n”},\ \text{“r”},\ \text{“o”}\}]),\\
\quad seq2 \in Seq([sn : (0 \,..\, SN\_MAX),\ tag : \{\text{“n”},\ \text{“r”},\ \text{“o”}\}]),\\
\quad val \in (0 \,..\, SN\_CNT),\ id \in networks] \triangleq\\
\quad\ \text{IF } Len(seq1) > Len(seq2) \text{ THEN}\\
\qquad \text{IF } Head(seq1)[TAG] = \text{“r”} \text{ THEN } \langle Head(seq1)\rangle \circ tag[Tail(seq1),\ seq2,\ val,\ id]\\
\qquad\ \text{ELSE } \langle [sn \mapsto Head(seq1)[SN],\ tag \mapsto \text{“o”}]\rangle \circ tag[Tail(seq1),\ seq2,\ val,\ id]\\
\quad\ \text{ELSE } \langle [sn \mapsto Head(seq1)[SN],\ tag \mapsto \text{“r”}]\rangle \circ Tail(seq1)
\end{array}
$$

Fig. 4. Marking algorithm in TLA$^+$

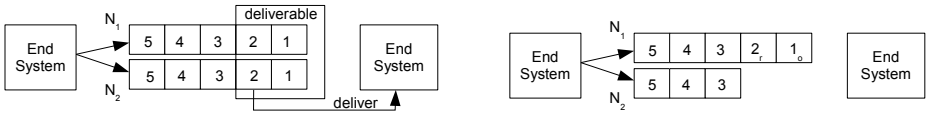

Fig. 5. The Marking Algorithm, with example behavior (left) and result (right)

network are considered to be deliverable to the receiving end system. When the second frame of N_2 gets delivered, the marking algorithm takes N_1 and N_2 without the first frame as input. As long as N_1 has more elements than the modified sequence N_2, the first element of N_1 is marked as old, if not already marked as redundant, and calls the function recursively with the tail of sequence N_1. If both sequences have equal length, the algorithm marks the first frame in N_1 as redundant and terminates, as can be seen in Figure 5. A full discussion of this algorithm and a correctness proof, based on Floyd's inductive assertion method for transition diagrams [4], can be found in [15].

4 The Properties

Traditionally an algorithm is examined to satisfy *safety* and *liveness* properties, stating that it behaves correctly and does not block. This is not appropriate in our situation as, before our formal investigation, we did not expect any of the redundancy management algorithms to be completely safe, which means neither forwarding redundant nor outdated frames to the application layer. The proposed algorithms in von Hanxleden [6] were not designed to satisfiy these safety requirements at all circumstances. That is why we refined the original properties relative to the behavior of the environment, e.g. one property allows the behavior to reset the sequence numbers, while a relaxed property would prevent such resets. This should help to distinguish the different algorithms and to decide about their quality. Besides safety and liveness a third class of requirements addresses properties regarding the *quality* of an algorithm. This includes the properties concerning the per frame loss as well as special scenarios with only one connected network. A fourth class of requirements, *availability*, addresses the behavior of the algorithms in case of network failures. As for the safety requirements, several refinement steps were defined to distinguish the analyzed algorithms. To give a complete description of all properties is beyond the scope of this paper, and we will restrict ourselves to give representatives of each of the four classes of properties. However, the numbering of the properties corresponds to the original thesis [15].

4.1 Safety

What does safety mean for RM? First of all, the RM shall not submit any redundant frames to the application layer. Secondly the RM shall preserve the order of frames and hence shall not submit *old* frames to the application layer. Each of these tasks can be weakened accordingly to the benignity of the environment.

***Redundancy 2*: If the environment does not reset anymore, the RM stabilizes and works properly from that time on.**

$$\begin{aligned} \mathit{Redundancy}2 \;\triangleq\; & \Diamond\Box\neg[\mathit{reset}]_v \\ & \Rightarrow \Diamond\Box(\forall\, \langle \mathit{id},\, \mathit{pos}\rangle \in \mathit{deliverable} : \\ & \text{ENABLED}\ \langle \mathit{extAcceptFrame}(\mathit{id},\, \mathit{env.frames}[\mathit{id}][\mathit{pos}][\mathit{SN}],\, \mathit{pos})\rangle_v \\ & \quad \Rightarrow \mathit{env.frames}[\mathit{id}][\mathit{pos}][\mathit{TAG}] \neq \text{“r”}) \end{aligned}$$

The premise of the TLA$^+$-formula expresses that from some state σ_i on, all consecutive states $\sigma_n \rightarrow \sigma_m$ are neither a *reset* nor a stuttering step. If this holds the redundancy management algorithm shall, after a finite time, only accept frames which are not marked as redundant.

***Order 1*: No old frame shall ever be submitted to the application layer.**

$$\mathit{Ordering}1 \;\triangleq\; \forall\, \mathit{frame} \in \mathit{out} : \mathit{frame}[\mathit{TAG}] \neq \text{“o”}$$

Obviously this is a very rigorous claim, which we expect to be failed by most of the proposed algorithms. Nevertheless this claim can be relaxed in the same way we relaxed the claim to never accept redundant frames to what is defined above in *Redundancy 2*.

4.2 Liveness

Of course the redundancy management algorithms shall not deadlock as long as it receives frames from its environment. More specifically:

***Liveness*: Each frame that is delivered to the RM will be either accepted or rejected.**

$$\begin{aligned} \mathit{Liveness} \;\triangleq\; & \forall\, \langle \mathit{id},\, \mathit{pos}\rangle \in \mathit{deliverable} : \\ & \vee\ \text{ENABLED}\ \langle \mathit{extAcceptFrame}(\mathit{id},\, \mathit{env.frames}[\mathit{id}][\mathit{pos}][\mathit{SN}],\, \mathit{pos})\rangle_v \\ & \vee\ \text{ENABLED}\ \langle \mathit{extRejectFrame}(\mathit{id},\, \mathit{env.frames}[\mathit{id}][\mathit{pos}][\mathit{SN}],\, \mathit{pos})\rangle_v \end{aligned}$$

It is expected that all algorithms will satisfy this property as they are all of type: IF *condition* = TRUE THEN *accept* ELSE *reject*, which implies that there exists a unique decision for each frame. That is why we do not mind that our formula is stronger than needed as this formula reasons about all frames and not only the set of received frames. It is enough to know that the *Liveness* formula implies absence of deadlocks.

4.3 Quality

One of the original requirements [6] states that *the redundancy management shall maintain the availability of a single network*. In other words, it shall not increase the number of frames lost that would be obtained with one of two networks normally running and

alone. This is a difficult, but important demand. Assume a fast but unreliable, hence lossy network A and a slower, completely reliable network B. It follows that an algorithm would need to use buffering to solve this problem. Buffering is considered harmful since it produces possibly large delays. So a gradation was introduced to obtain a more realistic estimation for the performance of the algorithms.

***Quality 0*: If only one network is connected to the RM, all received frames are forwarded.**

$$\begin{aligned} Quality0 \triangleq\ & \Box(\forall\, id1,\, id2 \in networks : isAlive[id1] \wedge \neg isAlive[id2]) \\ & \Rightarrow \Box(\forall \langle id,\, pos\rangle \in deliverable : \\ & \quad \neg\text{ENABLED}\ \langle extRejectFrame(id,\, env.frames[id][pos][SN],\, pos)\rangle_v) \end{aligned}$$

Hence, if the RM receives only frames from one network, the whole ES shall behave as with only one network running alone.

***Quality 1*: If both networks are alive and at least one member of a *Twin Frame* reaches the RM, one member gets submitted.**

$$\begin{aligned} Quality1 \triangleq\ & \forall\, id \in networks,\, pos \in (1\,..\,MCFL) : \\ & \wedge\, isAlive[id] \\ & \wedge\, isAlive[TNid[id]] \\ & \wedge \langle id,\, pos\rangle \in deliverable \\ & \wedge\, \text{ENABLED}\ \langle extRejectFrame(id,\, env.frames[id][pos][SN],\, pos)\rangle_v \\ & \Rightarrow \exists\, frame \in out : frame[SN] = env.frames[id][pos][SN] \end{aligned}$$

Quality 1 is equivalent to the original requirement, but it is easier to formalize that if a frame gets rejected, its twin frame has already successfully passed the redundancy management algorithm.

4.4 Availability

The goal of redundancy is to raise the availability of a system by duplicating parts of a system. In our case communication is done over multiple networks, since one single network is not reliable enough. The *Quality* requirements are concerned with the availability of the system in case of both networks operating. Now the case is considered that one network dies. This is the most important part of the properties. Redundancy is used to remain operating in presence of partial failures. These properties tell us how good an algorithm serves this task and finally enables a final decision whether this algorithm is a feasible choice.

***Avail 1*: If one network fails, all consecutive frames of the other, remaining network are accepted.**

$$\begin{aligned} Avail1 \triangleq\ & \exists\, id1,\, id2 \in networks : (isAlive[id1] \wedge \neg isAlive[id2]) \\ & \Rightarrow (\forall \langle id,\, pos\rangle \in deliverable : \\ & \quad \neg\text{ENABLED}\ \langle extRejectFrame(id,\, env.frames[id][pos][SN],\, pos)\rangle_v) \end{aligned}$$

An algorithm that satisfies this property would be a preferred choice (unless it fails all other requirements). Although this formula looks similar to *Quality 0* they describe different behaviors of the environment. While *Quality 0* specifies that from the beginning only one network is operating, *Avail 1* specifies that one network fails during execution.

Obviously an algorithm that handles absence of one network correctly at any time, it especially handles the situation where only one network operates from the beginning correctly. Hence *Avail 1* $\Rightarrow$ *Quality 0* holds. To satisfy this formula, it would help if the RM could detect whether a network is down; this task, however, was explicitly moved to the *Network Management*.

***Avail 2*: If one network fails and no *reset* occurs from that time on, all consecutive frames of the remaining network are accepted.**

$$
\begin{aligned}
Avail2 \;\triangleq\; & \Box(\, \wedge \Box(status \neq \text{“reject”}) \\
& \quad \wedge \exists\, id1,\, id2 \in networks : (isAlive[id1] \wedge \neg isAlive[id2]) \\
& \quad \Rightarrow (\forall \langle id,\, pos\rangle \in deliverable : \\
& \qquad \neg\text{ENABLED } \langle extRejectFrame(id,\, env.frames[id][pos][SN],\, pos)\rangle_v))
\end{aligned}
$$

5 Three Redundancy Management Algorithms

Of the thirteen RM algorithms considered in the original report [6], we now present a selection of three algorithms. As with the environment, the first step to model the algorithms behaviors is to define appropriate actions. This specification of actions is trivial, because all the RM can do is to receive a frame and decide whether it should submit or discard it.

5.1 RMA1

Recall that the *Sequence Number* (SN) of frame f is $SN(f) \in \{0 \ldots SN\_MAX\}$. The *Received Sequence Number* of frame f is denoted as $RSN(f)$ and it may be $SN(f) \neq RSN(f)$. However we assume those frames with $SN(f) \neq RSN(f)$ are not well-formed and thus discarded by the integrity checking. The *Previous Twin Network Frame* $PTN(f)$ for a frame f received on network N_1 denotes the last frame received on the other network N_2.

Definition 3 *(Sequence Number Skew) The Sequence Number Skew (SNS) of frame f is*

$$SNS(f) =_{def} RSN(f) -_{SN} RSN(PTN(f)).$$

This leads to the first RM algorithm considered here: **RMA1** specifies to *“accept a frame if and only if its sequence number skew is positive”*. The corresponding TLA$^+$ specification is given in Figure 6.

The formal analysis of RMA1 with TLC revealed that this algorithm works correctly if both networks are operational. However, if one network fails, RMA1 at some point starts rejecting frames from the remaining network due to the finite sequence numbering, as illustrated in Figure 7. This violates the property *Avail 2*. Moreover the algorithm will periodically discard non-redundant frames.

5.2 RMA3

As an evolution from RMA1, the next RM algorithm considered here compares not only the sequence numbers between frames of different networks, but also the difference of successive frames of the same network. Let $PAF(f)$ be the previously (from the RMA) accepted frame, prior to the reception of a frame f.

Accept frame IF frames are available AND $(SNS(f) > 0)$

$acceptFrame(id,\ sn) \triangleq$
$\quad \wedge\ snSkew[id,\ sn] > 0$
$\quad \wedge\ rm' = [rm \text{ EXCEPT } !.rsn = sn,\ !.ptn[id] = sn]$

Reject frame IF frames are available AND $SNS(f) <= 0$

$rejectFrame(id,\ sn) \triangleq$
$\quad \wedge\ snSkew[id,\ sn] \leq 0$
$\quad \wedge\ rm' = [rm \text{ EXCEPT } !.rsn = sn,\ !.ptn[id] = sn]$

Fig. 6. Specification of RMA1

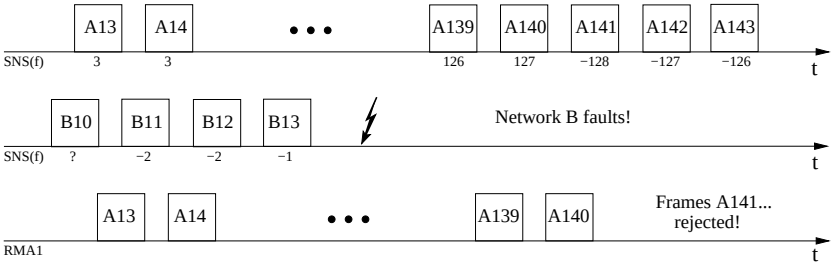

Fig. 7. Silent Network Scenario—Using RMA1. The upper time lines indicate frames received along the redundant network, along with their corresponding Sequence Number Skews. The lower time line indicates frames that RMA1 lets through to the application.

Definition 4 *(Previously Accepted Sequence Number) Let f be a frame with* $SN(f) \in \{0 \ldots SN\_MAX\}$*. Then the Previously Accepted Sequence Number (PASN) of f is given by*

$$PASN(f) =_{def} RSN(PAF(f)).$$

Definition 5 *(Sequence Number Offset) The Sequence Number Offset of a frame f is given by*

$$SNO(f) =_{def} RSN(f) -_{SN} PASN(f)$$

RMA3 specifies to *"accept if and only if the maximum of the sequence number skew and the sequence number offset is positive"*. The corresponding TLA$^+$ specifications to accept or reject the currently received frame are shown in Figure 8. Note how *paf* keeps track of accepted SNs. Model checking RMA3 proved that this algorithm will handle network failures better than RMA1. In case of a faulty network and no reset of the sequence number this algorithm will accept all subsequent frames. But as it turns out, RMA3 still needs two operating networks to cope with sequence number resets.

5.3 RMA13

The last algorithm considered here, RMA13, circumvents this limitation by introducing the concept of time. TLA$^+$ allows the modeling of continuous real-time aspects. Unfortunately, a first, very detailed timing model was not checkable with TLC, because

$$
\begin{array}{l}
\text{Accept frame IF frames are available AND } (SNS(f) > 0 \text{ OR } SNO(f) > 0) \\
acceptFrame(id,\ sn) \ \triangleq \\
\quad \land\ \lor\ snSkew[id,\ sn] > 0 \lor snOffset[sn]\ > 0 \\
\quad \land\ rm'\ = [rm \text{ EXCEPT } !.rsn = sn,\ !.paf = sn,\ !.ptn[id] = sn]
\end{array}
$$

$$
\begin{array}{l}
\text{Reject frame IF frames are available AND } SNS(f) < = 0 \text{ AND } SNO(f) < = 0 \\
rejectFrame(id,\ sn) \ \triangleq \\
\quad \land\ snSkew[id,\ sn] \leq 0\ \land\ snOffset[sn] \leq 0 \\
\quad \land\ rm' = [rm \text{ EXCEPT } !.rsn = sn,\ !.ptn[id] = sn]
\end{array}
$$

Fig. 8. Specification of RMA3

$$
\begin{array}{l}
\text{Exceed time bound:} \\
wait\ \triangleq\ \land\ rm.time = \text{TRUE} \land rm' = [rm \text{ EXCEPT } !.time = \text{FALSE}]
\end{array}
$$

$$
\begin{array}{l}
\text{Accept frame IF frames are available AND } SNS(f) > 0 \\
acceptFrame(id,\ sn) \ \triangleq \\
\quad \land\ \lor\ rm.pan = \text{“all”} \lor id = rm.pan \lor rm.time = \text{FALSE} \\
\quad \land\ rm'\ = [rm \text{ EXCEPT } !.pan = id,\ !.time = \text{TRUE}]
\end{array}
$$

$$
\begin{array}{l}
\text{Reject frame IF frames are available AND } SNS(f) < = 0 \\
rejectFrame(id,\ sn) \ \triangleq \\
\quad \land\ rm.pan \neq \text{“all”} \land id \neq rm.pan \land rm.time = \text{TRUE} \\
\quad \land\ rm' = [rm \text{ EXCEPT } !.time = \text{TRUE}]
\end{array}
$$

Fig. 9. Specification of RMA13

of state explosion. However, since the only real-time aspect in the redundancy management algorithms is a time-out, we could model this as a another ordinary action in TLA$^+$ that can occur under specific circumstances. **RMA13** is specified to *"accept if and only if the frame has the same network identifier as the last accepted frame or after time out"*. The corresponding TLA$^+$ specification is given in Figure 9. Note that in addition to the already known actions to accept or reject a frame, an action is introduced to model that the maximum time between two successive accepted frames is exceeded. RMA13 actually deviates from the *first valid wins* strategy, meaning that the first valid twin frame should always be passed to the application. Model checking revealed that, although RMA13 may cause a higher per frame loss than other algorithms, it behaves best in critical situations like network failures and sequence number resets. The most important advantage of RMA13 is that it satsfies all safety properties, which means that it never submits redundant or outdated frames to the application layer.

A complete and thorough analysis of the remaining algorithms would exceed the bounds of this paper. The interested reader can find the complete analysis in the thesis [15].

6 Assessment of the RMA Algorithms and their Formal Modeling

In summary, the formal analysis results indicate that one of the simplest proposed algorithms, RMA13, is the best choice overall. Although it does not increase the macro-

scopic availability, it satisfies all safety related properties, as it never submits redundant or outdated frames to the application, which is considered more important for the redundancy management task. Furthermore, the formal investigation of the RM task revealed that the decision to have a relatively small sequence number range (only 8 bits, as opposed to 28 bits, as had been considered originally) is not only economical, but prevents some catastrophic behaviors, too.

In our experience, TLA$^+$ is very suitable to specify the redundancy management algorithms. Though the compact notations of TLA$^+$ might be a little bit confusing at a first glance, they are very practical and maintain a good readability. Moreover the concept of untyped variables turned out to be not as error-prone as expected and is indeed very flexible.

The experience with the model checker TLC was also positive overall, but we did encounter some complications and at times strange or (we think) even faulty behaviors. TLC allows to check a wide range of expressible TLA$^+$ properties, however, temporal formulas that shall be checked with TLC and which contain actions must be of the form $\Box\Diamond A$ ("always eventually A") or $\Diamond\Box A$ ("eventually always A"), where A denotes the action. A work-around for this is to introduce another variable that records the action actually taken and to use this variable instead of actions in the formulas.

The possibility of constraining infinite models with specifying constraints on the defined variables is a major quality of TLC. Nevertheless the user must take care that the specified and checked invariants not only hold on the constrained state space. TLC checks for each state if all invariants hold and subsequently if all constraints are satisfied. Thus a state, which is considered unreachable may cause a violation of an invariant. This increases the complexity of optimizing the TLA$^+$ specifications for model checking with TLC.

A complete and more detailed description of the observed limitations can be found in the original thesis [15].

7 Summary and Conclusions

Of course, model checking a large set of algorithms and properties has its limitations. In our case, there is no way to give reliable conclusions of how many frames one algorithm accepts and rejects. However, the formal specification of the RM algorithms did detect gaps in the original specification. The specification of one algorithm turned out to be wrong, such that this algorithm failed all defined properties.

Writing formal specifications forces the engineer to clearly express what an algorithm must and what it must not do. The formalization effort presented here therefore focused on stating a precise set of requirements, and checking which algorithms fulfill which requirements. In contrast, the original, informal investigation followed an evolutionary, scenario-based approach; it started with a simple RM algorithm, detected scenarios where this RMA failed, and subsequently extended/modified the RMA. We have noticed that some RMAs fail the same properties, however, behave differently for certain scenarios. This suggests that the requirements in the formalization, which were already significantly refined compared to the original report, still could be refined further.

Acknowledgments

Eddie Gambardella has helped to formulate the redundancy management problem and to develop the original algorithms. We would also like to thank Leslie Lamport, for valuable discussions regarding TLA$^+$ and TLC, and the anonymous reviewers, for providing valuable feedback on this manuscript.

References

1. ALEXANDER, A. *Komposition Temporallogischer Spezifikationen - Spezifikation und Verifikation von Systemen mit Temporal Logic of Distributed Actions*. PhD thesis, Humboldt-Universität zu Berlin, 2005.
2. ARINC INCORPORATED. Homepage. `http://www.arinc.com`.
3. BREYER, R., AND RILEY, S. *Switched, Fast and Gigabit Ethernet*, third edition ed. MacMillan Technical Publishing, 1999.
4. FLOYD, R. W. Assigning meaning to programs. In *Proceedings AMS Symposium Applied Mathematics* (1967), pp. 19–31.
5. GORALSKI, W. *Introduction to ATM Networking*. McGraw-Hill, 1995.
6. HANXLEDEN, R. V., AND GAMBARDELLA, E. AFDX Redundancy Management, Feb. 2001.
7. LAMPORT, L. A summary of TLA+. `http://research.microsoft.com/users/lamport/tla/tla.html`.
8. LAMPORT, L. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems 16*, 3 (May 1994), 872–923.
9. LAMPORT, L. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002. `http://research.microsoft.com/users/lamport/tla/book.html`.
10. MANNA, Z., AND PNUELI, A. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
11. PNUELI, A. The temporal logic of programs. In *In Proceedings of the 18th Symposium on Foundations of Programming Semantics* (1977), pp. 46–57.
12. SINHA, P., AND SURI, N. On simplifying modular specification and verification of distributed protocols. In *HASE '01: The 6th IEEE International Symposium on High-Assurance Systems Engineering* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 173–181.
13. SOMMERFELD, L. Spezifikation eines 20 l-Perfusionsbioreaktor in TLA+. Diploma thesis, Universität Bielefeld, 1997.
14. TANENBAUM, A. S. *Computernetzwerke*, 4th edition ed. Prenctice Hall, 2003.
15. TÄUBRICH, J. Formal specification and analysis of a redundancy management system with TLA+. Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science, Mar. 2006. `http://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/jat-dt.pdf`.
16. US DEPARTMENT OF DEFENSE. Aircraft internal time division command/response multiplex data bus (mil-std-1553b). US Department of Defense, 1978-09-21. `http://dodssp.daps.dla.mil`.
17. YU, Y., MANOLIOS, P., AND LAMPORT, L. Model checking TLA$^+$ specifications. In *Proceedings of Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29* (1999), L. Pierre and T. Kropf, Eds., vol. 1703 of *Lecture Notes in Computer Science*, Springer, pp. 54–66. `http://link.springer.de/link/service/series/0558/bibs/1703/17030054.htm`.