April 2006

Formal Modeling and Analysis of AFDX Frame Management Design

Madhukar Anand
University of Pennsylvania, anandm@cis.upenn.edu

Samar Dajani-Brown
Honeywell Technology Center, samar.dajani-brown@honeywell.com

Steve Vestal
Honeywell Technology Center, Steve.Vestal@honeywell.com

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Recommended Citation

Madhukar Anand, Samar Dajani-Brown, Steve Vestal, and Insup Lee, "Formal Modeling and Analysis of AFDX Frame Management Design", . April 2006.

Formal Modeling and Analysis of AFDX Frame Management Design

Abstract

The Avionics Full Duplex Switched Ethernet (AFDX) has been developed to provide reliable data exchange with strong data transmission time guarantees in internal communication of the aircraft. The AFDX design is based on the principle of a switched network with physically redundant links to support availability and be tolerant to transmission and link failures in the network.

In this work, we develop a formal model of the AFDX frame management to ascertain the reliability properties of the design. To capture the precise temporal semantics, we model the system as a network of timed automata and use UPPAAL to model-check for the desired properties expressed in CTL. Our analysis indicates that the design of the AFDX frame management is vulnerable to faults such as network babbling which can trigger unwarranted system resets. We show that these problems can be alleviated by modifying the original design to include a priority queue at the receiver for storing the frames. We also suggest communicating redundant copies of the reset message to achieve tolerance to network babbling.

Formal Modeling and Analysis of the AFDX Frame Management Design [*]

Madhukar Anand
University of Pennsylvania,
Philadelphia, PA 19104
anandm@cis.upenn.edu

Steve Vestal
Honeywell Technology Center,
Minneapolis, MN 55418
Steve.Vestal@honeywell.com

Samar Dajani-Brown
Honeywell Technology Center,
Minneapolis, MN 55418
Samar.Dajani-Brown@honeywell.com

Insup Lee
University of Pennsylvania,
Philadelphia, PA 19104
lee@cis.upenn.edu

Abstract

The Avionics Full Duplex Switched Ethernet (AFDX) has been developed to provide reliable data exchange with strong data transmission time guarantees in internal communication of the aircraft. The AFDX design is based on the principle of a switched network with physically redundant links to support availability and be tolerant to transmission and link failures in the network.

In this work, we develop a formal model of the AFDX frame management to ascertain the reliability properties of the design. To capture the precise temporal semantics, we model the system as a network of timed automata and use UPPAAL *to model-check for the desired properties expressed in CTL. Our analysis indicates that the design of the AFDX frame management is vulnerable to faults such as network babbling which can trigger unwarranted system resets. We show that these problems can be alleviated by modifying the original design to include a priority queue at the receiver for storing the frames. We also suggest communicating redundant copies of the reset message to achieve tolerance to network babbling.*

1 Introduction

Control systems in general and avionics systems in particular, rely on complete and up to date data delivered from the source to receiver in a timely fashion. For safety-critical systems, reliable real-time communication links are essential. The Avionics Full Duplex Switched Ethernet (AFDX) [3], has been developed to meet these requirements for commercial aircraft applications. The AFDX is a subset of the profiled version of IEEE 802.3 standard Ethernet [9], with key enhancements to provide deterministic timing and reliable delivery of messages. Deterministic timing is achieved through communication over virtual links (VL) that have a bounded bandwidth and frame delivery interval. Communication over redundant channels is used to achieve reliable delivery of the messages. A frame management mechanism is responsible for checking integrity of message frames and managing the redundancy before delivering the messages to the application. Therefore, the frame management forms an important component of the AFDX design and has to be guaranteed against design flaws.

In this work, we develop a formal model of the AFDX frame management, analyze and verify whether it meets the requirement specification under different kinds of network faults. In developing a formal model of the frame management, we use timed automata [1] that can quantitatively capture the temporal information. Our specific model consists of a network of timed automata with a transmitting end system, two communication channels and a receiving end system. The system is described in UPPAAL [5] which supports model-checking properties specified in CTL. From our analysis, the design was found to be vulnerable to faults like network babble which led to unwarranted resets and dropped frames if they arrived out-of-order. To fully utilize the redundancy in messages and use this redundancy to detect faults, we propose including a priority queue at the receiver. This will help detect network babble on a channel, and deliver frames in sequence to the application even if they arrive out-of-order. To reduce the probability of erroneous resets, we suggest communicating redundant copies of the reset message. These modifications can easily be incorporated into the original design and provide increased reliability to the AFDX frame management.

The remainder of this paper is organized as follows: Sec-

[*] This research was supported in part by NSF CCR-0209024 and ARO DAAD19-01-1-0473.

tion 3 introduces the AFDX and the frame management, Section 4 describes the timed automata model of the system, Section 5 presents the analysis and results. Finally, we present the modifications in Section 6 and conclude in Section 7.

2 Previous Work

The ARINC-664 [2] is a commercial standard for the avionics communication architecture. The AFDX [3] is a vendor specific implementation of this standard. It is based on the 802.3 standard Ethernet with enhancements to ensure determinism and reliability. An overview of a switched Ethernet avionics network along with testing challenges are identified in [10]. While their work concentrates on hardware testing of various modules through simulation, our focus here, is to formally model and analyze the design under different faults. Our model was developed using UPPAAL, a freely available tool that allows modeling with a flavor of timed automata, called the Timed Safety Automata [7]. The modeling language in UPPAAL builds on the timed automata model, providing useful extensions like integer variables, broadcast channels, urgent and committed locations, etc. The model-checker in UPPAAL allows specifying queries using a simplified version of CTL, where the query language consists of path and state formula, but the path formulae cannot be nested. A significant body of literature exists on modeling and verification of protocols in UPPAAL(c.f., [6, 8, 12]).

3 AFDX and the Frame Management

The main elements of the AFDX network are end-systems, switches, and links. The function of the End-system (ES) is to provide services, which guarantee a secure and reliable data exchange to the partition software. Each end-system has a direct, bidirectional connection to a switch. There may be multiple such connections to be used for redundant communication. The switched network ensures that the connection and bandwidth required to move data from one end-system to another is available. Quality of Service (QoS) provides a method for categorizing traffic and for ensuring that particular categories of traffic will always flow across the network at the service level to which they are entitled, regardless of competing demands. For the aircraft network, each network transmission request must be serviced regardless of the data type and a maximum network transit delay, called end-to-end latency(L), must be guaranteed. A guaranteed service provides a firm, mathematically provable, upper bound on end-to-end latency.

The Virtual Link(VL) is the basis of the Ethernet protocol. Each VL defines an unidirectional connection from one source end-system to potentially multiple destination end-systems through which data frames are exchanged. The data flow on the links is controlled by the end-system in accordance with a Bandwidth Allocation GAP(BAG). The BAG values are time slices allocated by and end-system to transmit data for a VL. These times are defined in milliseconds and are typically powers of two.s

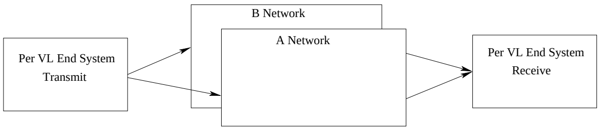

Figure 1. Network Redundancy Concept

Reliable frame delivery in the AFDX design is ensured by utilizing redundant links. This basic idea of network redundancy is shown in Figure 3. End-systems communicate over multiple communication channels with the effect that communication is protected against loss of one complete network.

The redundancy scheme operates on a per link basis in the following manner: A transmitting end-system prepares some data and passes it to the communications protocol stack. Here, a sequence number field is added to each frame to enable the receive function to reconstruct a single ordered stream of frames without duplication before delivery to the receiving partition. The sequence numbers are one octet long with a range from 0 to 255 and are incremented on each successive frame. After 255, the sequence number is wrapped around to 1. The sequence number 0 is reserved for communicating resets. In this way the partition is unaware of the underlying network redundancy, and a simple interface can be built between the communications stack and partitions that utilize the network service.

In the default mode, each frame is sent across both of the networks and the redundancy is taken care at the receiving end-system. In order to simplify the algorithm at the receiving end-system, redundant copies of a frame should be sent within a maximum time difference of 0.5 ms. Upon reception, an algorithm in the communications stack (below IP layer) uses a “First Valid Wins” policy. This means that the first frame to be received from either network with the next valid sequence number is accepted and passed up the stack to the receiving partition. When the second frame is received with this sequence number, it is simply discarded. As the flow of frames given in Figure 3 below indicates, Redundancy Management (RM) is placed after the Integrity Checking (IC).

Under fault-free network operation, the IC simply passes the frames that it has received on to the RM, independently for each network. If there are faults (based on sequence number), the IC has the task of eliminating invalid frames, and informs the network management accordingly. For each

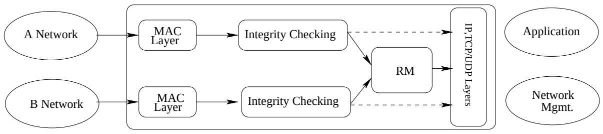

Figure 2. Network Redundancy Concept

network the IC tests each frame for a sequence number in the interval: $[PSN \oplus 1, PSN \oplus 2]$ where Previous Sequence Number (PSN) is the sequence number of the previous frame received (but not necessarily forwarded) on this VL. The operator $\oplus$ takes the wrap-around of sequence numbers into account. So, for example if $PSN = 254$, then $PSN \oplus 1 = 255$ and $PSN \oplus 2 = 1$.

The function of the AFDX redundancy management is merely to eliminate frames that are redundant copies of frames that it has already passed on to the partition.The RM assumes that the network is working properly and, in particular, the deterministic properties are verified. RM configuration is generally based on the $SkewMax$ parameter: i.e. the maximum time between the reception of two redundant frames. This value depends on the network topology (number of switches crossed by the frames) and should be provided by the system integrator. The $SkewMax$ value (expressed in ms) is given by configuration per VL.

4 System Model

We model the AFDX frame management, introduced in the previous section as a network of timed automata. The timed automata model allows us to quantitatively capture the temporal aspects of the frame management such as maximum latency, skew and the BAG. The model was developed using UPPAAL.In our model, we have three principals : The transmitting end-system, the channel, and the receiving end system. We will restrict ourselves to the case of two redundant channels. However, extending the reasoning for more redundant channels is fairly straightforward.

4.1 Transmitting End-system

The transmitting end-system sends the messages on the redundant channels. The message (`msg`) is assumed to be broken into frames (`fr`) that are then communicated across the channels. The model for the transmitting system is given in Figure 3. The initial state is `Init` from which the messages are sent on both the channels within 0.5ms of each other. Since UPPAAL allows only integer constraints on clock variables, in the model, we use `c<1` to capture this constraint.

The actual sending of the messages is captured via the channels `msg_one` and `msg_two`. After the transmission, the system waits till the end of the BAG to transit back to the initial state. From the initial state, the system can non-deterministically progress to the `Reset` state that captures the reset of the transmitting end-system. If there is a reset, then we wait for time `HR` before sending frame 0, indicating a reset, on both the channels. This reflects the time the transmitting system takes to go though an hardware reset. In the model, a boolean variable `rchk` is set whenever the system is reset. This will help us to trace the execution of a reset and also ascertain whether the receiving system resets in response to a transmitter reset.

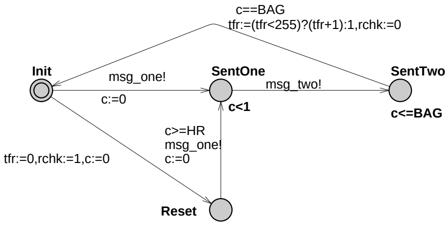

Figure 3. Timed Automata Model of the Transmitting End-system

4.2 Network Channel

The network channel is responsible for transmitting frames from the receiving end-system and deliver it to the receiving end-system. The focus of our work here is to model different kinds of network faults and view its impact on the frame management. In this model, we consider two kinds of network faults :

1. *Transmission Related:* Under this category, we consider errors such as bit-errors, dropped packets, etc. We model these errors as being non-deterministic and independent. In practice, however, it is commonly assumed that the probability of error in consecutive frames is close to zero. Nevertheless, assuming the errors to be independent keeps the model simple while retaining its implications in practice.

2. *Network Babble:* The network can sometimes babble i.e., deliver arbitrary frames to the receiver and we model this fault in the network channel. Again, we assume that the babbling is non-deterministic and independent of other faults.

Initially, the system is in the state `Idle`. Upon receiving the message `msgr` (which could be either `msg_one` or `msg_two`), it transits to the transmitting state. If the transmission is successful, then, `SendSuccessful` state is reached. The system can remain in this state for as long

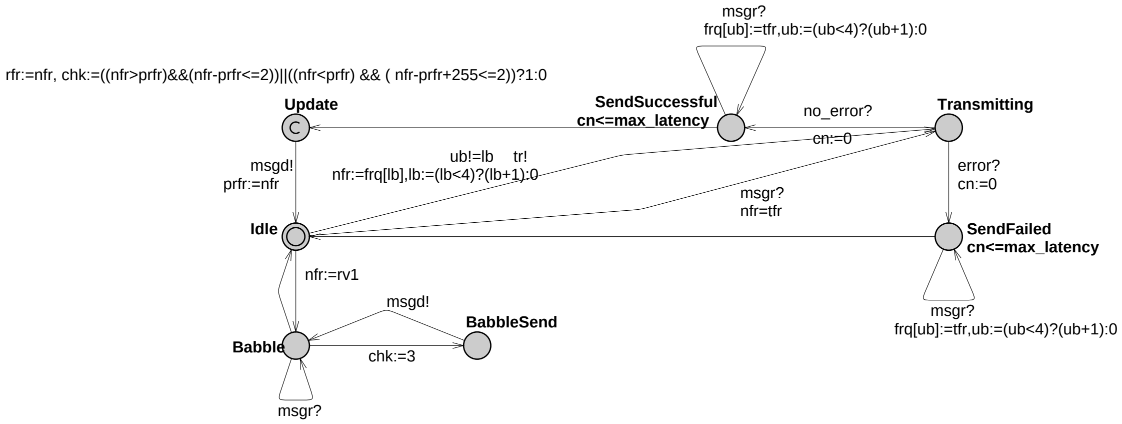

Figure 4. Timed Automata Model of the Channel

as `max_latency` and then the frame is delivered to the receiving end-system by passing through the `Update` state. The delivery of the frame is signaled by `msgd`. If the transmission fails, then `SendFailed` state is reached from which the system returns to the initial state. Network babbling is modeled in states `Babble` and `BabbleSend` during which the network delivers random numbered frames (`rv1`) to the receiving end-system.

We also implement a message frame queue in `frq`. In the model, we consider the maximum length of this queue to be 5. This allows the network and the transmitter to be independent. Whenever the the transmitting end-system transmits, the frame is queued and subsequently transmitted by the network.

A variable `chk` is introduced to keep track of frame being delivered. If a valid frame is being delivered, then, it takes a value 1 else it is assigned 0. If the frame being delivered is a result of babbling, then `chk` is assigned 3.

4.3 Receiving End-system

The receiving end-system implements the integrity checking (IC) and redundancy management (RM). Frames that are not in the interval $[PSN \oplus 1, PSN \oplus 2]$ are discarded by the IC, except in the case when the frame number is 0. Frames may also be discarded in the RM because of the "first valid wins" policy. Both these policies are implemented in our model.

The automata for receiving end-system, given in Figure 5, is initially in the state `NotRecd`. If a valid packet is received from the first channel, then it updates the previous sequence number (`psn`) by going through the state `RecdOne`. A valid frame is similarly handled through the state `RecdTwo`. Both these states are labeled as committed. Therefore, time is not allowed to pass in this state and the transition back to `NotRecd` is taken immediately. `psnlb` and `psnub` are used to keep track of the interval bounds. Upon receiving a 0, the system is reset via the state `Reset`.

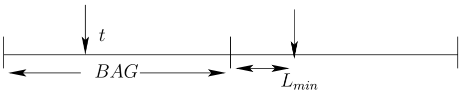

Figure 6. A system with two agents

We incorporate several variables to check for certain cases of interest such as acceptance of a babbling frame, resets, etc.:

1. `chk`: This variable is shared with the network channel and is assigned 2 when it is discarded by the RM but passes through the IC.

2. `resetchk` : This is a boolean variable that is set whenever the receiving system resets.

3. `babchk` :A boolean variable that is set whenever a babbling frame is accepted. Note that the babbling frame is tracked by verifying if `chk=3`.

4. `valid` : This variable is also a boolean that is set whenever the frame is accepted by entering one of the states `RecdOne` or `RecdTwo`.

5 Analysis and Results

The semantics of the frame management depend on the relationship between various parameters such as, actual latency, the skew between reception of redundant frames, and the time for hardware reset. Here, we consider two distinct cases:

Let L_{max} be the maximum latency of arrival of a frame, L_{min} the minimum latency, and $SkewMax,$ the time between delivery of redundant frames. Consider the scenario

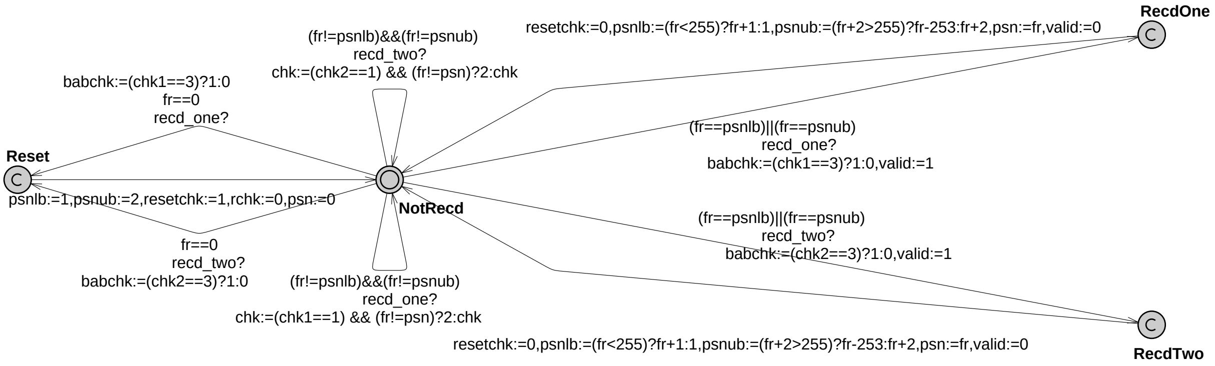
RecdOne
resetchk:=0,psnlb:=(fr<255)?fr+1:1,psnub:=(fr+2>255)?fr-253:fr+2,psn:=fr,valid:=0
(fr!=psnlb)&&(fr!=psnub)
recd_two?
chk:=(chk2==1) && (fr!=psn)?2:chk
babchk:=(chk1==3)?1:0
fr==0
recd_one?
Reset
(fr==psnlb)||(fr==psnub)
recd_one?
babchk:=(chk1==3)?1:0,valid:=1
psnlb:=1,psnub:=2,resetchk:=1,rchk:=0,psn:=0
NotRecd
(fr==psnlb)||(fr==psnub)
recd_two?
babchk:=(chk2==3)?1:0,valid:=1
fr==0
recd_two?
babchk:=(chk2==3)?1:0
(fr!=psnlb)&&(fr!=psnub)
recd_one?
chk:=(chk1==1) && (fr!=psn)?2:chk
resetchk:=0,psnlb:=(fr<255)?fr+1:1,psnub:=(fr+2>255)?fr-253:fr+2,psn:=fr,valid:=0
RecdTwo

Figure 5. Timed Automata Model of the Receiving End-system

of frame arrivals as shown in Figure 6. Let us assume that a frame was delivered at time t on the first channel. The redundant frame on the other channel can arrive as late as $t + SkewMax$ and the successive frame on the first channel may arrive as early as L_{min} in the next BAG. Therefore if $BAG - t + L_{min} > SkewMax$, then, the redundant frame will arrive before the successive frame on the first channel. Since t can be as large as L_{max} in the worst case, to require that redundant frames arrive before the successive frame, we should have $BAG - L_{max} + L_{min} > SkewMax$. We now analyze the model based on this condition.

1. Case $SkewMax < (L_{max} + BAG - L_{min})$:
   This situation may apply to many types of messages such as those with real-time data. In this case, although the frames arrive in-order, we would want to test the behavior under transmission faults, network babble and reset messages.

   (a) *A babbling frame is never accepted* : The variable `babchk` was used to verify this property. The desired property is the expressed by the CTL condition `E<> (rs1.babchk && rs1.valid)`. This property was satisfied in the model, and the following diagnostic trace was generated: If one of the network babbles such that the babbling frame number lies in $[PSN \oplus 1, PSN \oplus 2]$, then this gets accepted and in this process, the legitimate frame from the other network gets rejected even though it is delivered successfully.

   (b) *A receiving end-system reset implies a transmitting system reset*: This was expressed using the condition, `A[] rs1.Reset imply rchk`. The following counterexample was generated in this case: If a network babbles a reset frame number, then that results in the receiving end-system erroneously reseting.

2. Case $SkewMax \geq (L_{max} + BAG - L_{min})$:
   For certain kind of messages, such as those with multimedia content, the skew may actually be longer than $L_{min} + BAG - L_{max}$. In this case, it is possible that, the redundant frames arrive after the next frame arrives on the first channel. Therefore, apart from ensuring that babbling frames are never accepted and no erroneous resets, we test whether a frame is ever dropped when delivered to the receiver.

   When model-checked, a counterexample was generated for all the three cases. We present the counterexample that was generated when testing for the last property. The counterexample for first two cases are similar to those given above.

   - *If a valid and non-redundant frame is delivered, then it is not discarded* This property was expressed using the CTL condition `A[] chk!=2`. When the faster of the two networks delivers successive invalid and valid frames before the slower network can deliver the first frame, the receiving end-system accepts the second frame from the faster network and considers the valid first frame as invalid and discards it.

     We note that although this scenario is mentioned in the AFDX design document [3], it has not been addressed there. Dropped frames could affect the QoS and hence should be avoided. In the next section, we show that a minor modification to the original design can help avoid this problem and thereby ensure good QoS.

6 Improving the Frame Management Design

The frame management design could be modified to be handle network babbling and also not disregard valid frames that arrive late. We suggest two changes to the design that will help us achieve these goals.

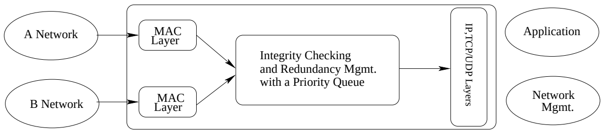

Figure 7. Redundancy Management with a Priority Queue

6.1 Integrating the IC and RM with a Priority Queue

The first suggestion for improving the frame management is the introduction of a integrity check with queuing module instead of the distinct IC and RM modules in the original design. The modified design is presented in Figure 7. Listing 1 describes the action taken when a frame `fr` with frame number `frn` is delivered to the module. The main idea is that, whenever a frame is delivered such that it is valid for that particular channel, it is enqueued. If that particular frame is already present in the queue, it is verified for consistency. If a channel delivers a frame with the same frame number twice, it is treated as network babble and the frame ignored. Since the previously delivered frame from the same channel could also be due to the network babble, we should delete it from the queue to avoid accepting frames that are not legitimate.

In the Listing 1, we assume that `enqueue` and `isPresent` are functions that are implemented. `enqueue` is assumed to take the message and the frame number as input and enqueue it in the queue according to the priority. `isPresent` is assumed to check whether a particular frame is already present in the queue and return the frame if present else return `null`. The case of resets i.e., `frn=0` is treated in the next section. We also assume that we have two counters, `psn[ch],ch=1,2`, that keep a track of the previous sequence number of that channel.

```
$wait(fr)$
if $((frn > 0) \wedge (frn \in [psn[ch] \oplus 1, psn[ch] \oplus 2]))$ then
    if $(isPresent(frn) = null)$ then
        enqueue(fr)
        $psn[ch] \leftarrow frn$
    else if $(isPresent(frn) \neq fr)$ then
        // Inconsistent frames
    end if
else if $(frn = psn[ch])$ then
    // Channel ch babbling
end if
```

Listing 1: Handling frames with the priority queue

Dequeuing the frames: Messages in the queue can be dequeued and handed over to the application at hand. Every time we enqueue an element, it can be timestamped and dequeued either upon receipt of the redundant frame from the other channel or after time `SkewMax`. If the redundant frame fails to arrive, then, we would have to wait for the successive frame to arrive on the channel, so that it can be tested for babbling (a wait of $L_{max} + BAG - Lmin$). The queue is also to be emptied after the frame 255 is received to keep it from interfering with the priorities once the frame number starts over from 1.

Compared to the modified design, where the additional latency could be as high as $L_{max} + BAG - L_{min}$, the original design introduces, at worst, a delay of of $SkewMax$ when the frame is dropped on the first channel. However, despite higher latencies, the modified design may be preferred for implementation as it offers better data integrity and QoS.

6.2 Handling Resets

One of the problems with the AFDX frame management is that it is vulnerable to babbling resets. The IC works as long as frame numbers are greater than 0, but accepts the frame 0 and resets. The problem is that, with just two channels and one reset message, we cannot achieve tolerance to both network babble and transmission loss: If only one reset message is received at the end-system, it could either be due to network babble on one channel or due to transmission loss on the other.

One strategy to achieve tolerance to babbling, is to increase the number of redundant channels. That way, a voting scheme [4] can be used to decide on resets. To achieve tolerance to one channel babble, we would need at least three channels. However, including more redundant channels adds a significant overhead and therefore not an attractive option for implementation.

An alternative to having more redundant channels, is to send redundant messages on each of the channels. The idea is that, instead of sending frame 0 once, we send two reset messages (frame 0) on both the channels. The receiver resets only when it gets at least one 0 frame from either of the channels. This is described in Listing 2. It can easily be seen that this modification makes the design tolerant to babbling on one channel and one message loss due to transmission. The disadvantage here, when compared to adding an extra channel, is the increased delay before the receiver reset. However, since there is no extra overhead, this scheme may be preferred over adding an extra channel.

Analysis on the modified design: We modified the timed automata models with the above suggestions and checked for the desired properties. Only the reset property `A[] rs1.Reset imply rchk` generated a counter example when both the frames on one channel were dropped. Although this is a possibility, the probability of this happening

```
$resetCounter[1] \leftarrow 0$
$resetCounter[2] \leftarrow 0$
$wait(fr)$
if $(frn = 0)$ then
    $resetCounter[ch] \leftarrow resetCounter[ch] + 1$
end if
if $(resetCounter[1] \geq 1) \wedge (resetCounter[2] \geq 1)$ then
    // reset
end if
```

Listing 2: Handling resets

in practice are extremely small.

6.3 Design Tradeoffs

We notice that there is a tradeoff between latency at the receiver versus adding an extra channel: By increasing the number of independent channels, we can reduce the latency and by reducing the number of channels, we would have to transmit multiple copies, and this results in increased latency. Therefore the exact design choice would have to depend on the application at hand. If the extra latency is acceptable, then we could do without an extra channel. However, many critical applications an extra channel would have to be employed to achieve fault tolerance.

7 Conclusions and Future Work

The AFDX is an implementation of the commercial standard for avionics communication architecture, called ARINC-664. It has been developed for providing reliable and deterministic delivery of frames in a switched Ethernet for avionics applications. To provide these guarantees, frames are sent over redundant links that have a bounded latency and bandwidth. The frame management is responsible for managing the redundancy and checking integrity of frames before handing it to the application. The frame management also aims to achieve tolerance to faults such as transmission errors and network babbling. In this work, we have developed a formal model of the AFDX frame management using a network of timed automata. From our analysis, we have uncovered that the design is vulnerable to babbling resets and dropping of frames. To address these issues, we have proposed integrating the redundancy management and integrity checking with the help of a priority queue, and duplication of reset message on each channel. These modifications are relatively simple to incorporate into the original design and help achieve tolerance to channel babble and a better QoS.

7.1 Testing AFDX Implementations

As future work, we propose to generate test suites based on the UPPAAL model we have developed. Testing AFDX implementations would involve injecting faults in the network as per the automata model and observing for changes registered at the receiving end-system. The faults injected in the network would then have to be controlled and the behavior observed for possibilities of different errors, including multiple instances of the same error and simultaneous occurrence of distinct types of errors. Online testing based on UPPAAL models have been developed [11] and we hope to adapt it for generating tests for our models.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] ARINC. Specification 664: Aircraft data network, part 7-deterministic networks(draft 2,oct 10,2003).

[3] ARINC. Arinc project paper 664: Aircraft data network, part 7-avionics full duplex switched ethernet(afdx) network, 2005.

[4] T. A.S and S. M.V. *Distributed systems:Principles and Paradigms*. Prentice Hall PTR, Uppersaddle River,NJ, USA, 2001.

[5] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In *4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236, September 2004.

[6] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Automated analysis of an audio control protocol using UPPAAL. *Journal of Logic and Algebraic Programming*, 52–53:163–181, July-August 2002.

[7] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lecture Notes on Concurrency and Petri Nets. W. Reisig and G. Rozenberg (eds.)*, number 3098 in LNCS. Springer–Verlag, 2004.

[8] A. David and W. Yi. Modelling and analysis of a commercial field bus protocol. In *Proceedings of the 12th Euromicro Conference on Real Time Systems*, pages 165–172. IEEE Computer Society, 2000.

[9] IEEE. Std.802.3:information technology, 1998.

[10] B. K and T. T. Switched ethernet testing for avionics applications. In *Proceedings of IEEE Systems Readiness Technology Conference*, pages 546–550, 2003.

[11] K. G. Larsen, M. Mikucionis, B. Nielsen, and A. Skou. Testing real-time embedded software using uppaal-tron: an industrial case study. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 299–306, New York, NY, USA, 2005. ACM Press.

[12] H. Lonn and P. Pettersson. Formal Verification of a TDMA Protocol Startup Mechanism. In *Proc. of the Pacific Rim Int. Symp. on Fault-Tolerant Systems*, pages 235–242, Dec. 1997.