# Formal Framework for Safety, Security, and Availability of Aircraft Communication Networks

Rohit Dureja*, and Kristin Yvonne Rozier†
*Iowa State University, Ames, IA 50011, USA*

**As the costs of fuel and maintenance increase and regulations on weight and environmental impact tighten, there is an increasing push to transition on-board aircraft networks to wireless, reducing weight, fuel, maintenance time, and pollution. We outline a candidate short-range wireless network for aircraft on-board communications using the common ZigBee protocol and privacy-preserving search implemented as a secure publish/subscribe system using specially coded meta-data. Formally specifying safety and security properties and modeling the network in NuXmv enables verification and fault analysis via model checking and lays the groundwork for future certification avenues. We report on experiments formally analyzing our candidate wireless network, showing overhead and availability for encrypted and fault-tolerant communications. We propose a formal model that allows system designers to estimate communication failure rates, and directly trade off fault-tolerance for bandwidth, while preserving communication security.**

## I. Introduction

Increasing system complexity and limitations of wired networks in avionics systems are driving innovations in wireless technology integration. This migration of wired networks to wireless is motivated by several advantages: wireless networks weigh less than wired networks, scale better than comparable wired networks, and are easier to install and maintain since they are essentially plug-and-play and don't require inspection of lengthy stretches of cable. Although the cost of equipment for wireless networks are higher by comparison, these costs are typically outweighed by the benefits of ease-of-use, scalability, lower weight, and lower setup-cost than a comparable wired network. However, there are some draw-backs to migrating from wired to wireless networks. For example, wireless networks are more vulnerable to cyber-security attacks, providing a larger attack surface. Additionally, the added complexity in protocol and organization of wireless networks increases their susceptibility to faults. We report findings from our recent work designing next-generation aerospace standards for efficient, reconfigurable, aerospace system networks. We examine the problem of selecting a wireless-enabled on-board network configuration along two dimensions: verification and formal fault analysis of a prototype wireless network using an off-the-shelf protocol, and investigation of overhead and network

*Graduate Student, Department of Computer Science, Iowa State University.
†Assistant Professor, Departments of Aerospace Engineering, Computer Science, Electrical and Computer Engineering, and Mathematics, Iowa State University.

availability for encrypted, fault-tolerant protocols for such networks.

A modern Boeing 787 has ~500km of wires; together with their harnesses they weigh nearly 7,400 kg, which is about 3% of the total weight of the aircraft [1]. An Airbus A380 also has ~100,000 wires totaling 470 km and weighing 5,700 kg; another ~1/3 of that weight is required for harnesses to hold these wires. Aerospace America describes this problem as the "War on Wiring," and estimates that up to 1,800kg of wiring could be removed from such aircraft, starting with non-avionics networks like passenger entertainment systems, system health management data networks, sensor networks, and networks carrying commands for flight-control [1]. Identifying wired components on the aircraft that can be migrated to wireless protocols while maintaining the robustness standards needed for flight certification remains an open question.

Migration of wired networks to wireless networks requires a thorough study of the wired system, choosing an appropriate wireless communication protocol, assessing faults and security concerns associated with the wireless communications carried on that network, and analysis of the quantitative benefits of using a secured or unsecured wireless network. Each of these tasks are interdependent and require substantial understanding regarding expectations from the introduced wireless network with the primary requirement being that *the new, wireless network needs to be at least as safe, and secure, as the existing wired network*. Our formal analysis framework enables analysis of wireless network models in terms of fault tolerance and fault propagation. A preliminary version of the framework appeared in [2]. This extended version improves the framework along several dimensions as detailed below.

**Contributions.** To the best of our knowledge, this is the first work that addresses the problem of communication technology migration in terms of formal system safety and fault tolerance. The formal methodology we present aids system designers in the comparison of different communication networks and the exploration of viable fault-tolerant mechanisms. The presented methodology builds upon existing model checking and safety assessment tools, and is plug-and-play, making it fully compatible with commercial-off-the-shelf (COTS) components. As a proof of concept, we formally model the ZigBee protocol and demonstrate analysis of a wireless network using ZigBee for its wireless protocol. Moreover, we propose additions to the ZigBee protocol that enhance the reliability and trustworthiness of wireless communication, while ensuring real-time deadlines are met. The new format adheres to existing ZigBee standards, and can be used with commercial equipment. These modifications give the wireless system the ability to alter fault-tolerance and throughput capabilities in response to changing conditions on the aircraft.

**Structure.** The rest of the paper is organized as follows. We highlight related work on formal fault analysis and fault tolerance in Section II. Section III details background information about model-checking, linear temporal logic (LTL), contract-based design, formal fault analysis, fault tolerance, and the ZigBee protocol. Sections IV and V form the main contribution of our paper: enhancements to the ZigBee protocol to improve fault tolerance, and formal modeling and

analysis of a prototypical ZigBee network. The experiments in Section VI validate the utility of our formal framework, and Section VII concludes by highlighting future work and possible extensions.

## II. Related Work

Previous work on fault-tolerant mechanisms for ZigBee wireless sensor networks focuses on using existing wireless mechanisms such as Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) [3], chained ZigBee repeaters, and the creation of multiple parallel transmission streams of the same data. These and similar mechanisms provide the basic tools for establishing safe ZigBee network topologies with the current state of the art. In the case of hardware failure, redundant networks of ZigBee transmitters have demonstrated fault-tolerance using fault detection and recovery mechanisms through disconnection and network re-join with altered node roles [4]. While these approaches have lead to more reliable systems, they do not account for the challenges of network migration. Wireless avionics, and human interfaces for spacecraft [5] are pushing towards migration of wired networks to wireless. In the absence of fault-tolerance mechanisms, experimental performance evaluation has been employed to assess network reliability [6, 7] to understand the likelihood of failure when it cannot be prevented. However, such experimentation is limited in terms of scenarios explored and environmental conditions, and does not scale. **Our network formal model focuses on inter-component communication rather than protocol behavior.** It allows the comparison of different networks in terms of safety, and is fully automatic and scalable.

Algorithms and techniques for searchable encryption [8–10] have been studied extensively in the context of cryptography, but have focused mostly on traditional cryptographic applications. The primary focus has been efficiency improvement and security formalization. The methods underlying searchable encryption [8] proved impractical and eroded security due to the necessity of generating every possible key that the search expression can match. To reduce the search cost, Bloom filters are used to create per-file searchable indexes [9] on the source data. However these studies required exact keyword search. In theory, the use of flexible search over regular expressions is allowed [11] but again the results prove impractical. Other recent studies have focused on cloud applications of data storage, such as the problem of similarity search [12] over the outsourced encrypted Cloud data and work that provides approximate search capability [13] for encrypted Cloud data.

Other studies in the literature make use of encryption techniques that ensure that user privacy is not compromised by a data center [14]. The challenge changes in these cases: the problem of how an encrypted database can be queried without explicitly decrypting the records. At a high level, a searchable encryption scheme [15] provides a way to encrypt a search index such that its contents are hidden except to a party that is given appropriate tokens. Public Key Encryption with keyword search [16] indexes encrypted documents using keywords. Public-key systems that support equality ($q = a$), comparison queries ($q > a$) as well as more general queries such as subset queries ($q \in S$) provide massive improvement for search-ability. A symmetric cryptography setting for searchable encryption architectures [8]

for equality tests advanced the state-of-the-art.

## III. Background

### A. Model Checking

Model checking has been successfully used in the aviation industry to verify that avionics systems uphold their safety requirements [17–31]. We choose model checking as a verification method for our study due to the unique properties provided by this verification technique over using simulation or testing. These include the following:

- Model checking is an exhaustive analysis of the entire behavior space of the system, for all possible inputs; it is not probabilistic.
- Model checking can analyze partial or incomplete designs, i.e., by exhaustively exploring all possible paths for the unspecified design elements. It can analyze both a logical system (such as code or a hardware logic design) and models of such a design as long as they are modeled in a formal semantics. This flexibility is required for our wireless network analysis, as we need to analyze a model of the partial design. (If safety requirements are violated by an exhaustive check of all possible extensions of a partial design, adding more detail or implementation will not save the system.)
- The exhaustive nature of model checking means that we can prove the *absence* of bad behaviors in addition to the presence of good behaviors; these two sides of the proof are required to construct an appropriate safety case for eventual flight certification, e.g., as required by DO-178B [32], DO-178C [33], DO-333 [34], and DO-254 [35].
- Model checking finds the existence of any "bad" system execution, not how often that path is explored; in other words, for this type of analysis all bad paths are of equal weight. This is important for our early-design-time analysis because we want to be able to efficiently find violations of our safety requirements, however rare, so that we can proceed with a more complete knowledge of what could go wrong than we can obtain, e,g., from rare-event simulation.

Tools based on model-checking technology [36, 37] have enjoyed a substantial and growing use over the last few years, and have recently been used to comparatively analyze multiple possible avionics system designs to narrow the design space early in the system design process [20, 21, 38]. Given a system or system model $M$ in some formal semantics, and a requirement $\varphi$ in some mathematical logic, model checking is the task of exhaustively and automatically checking whether the model satisfies the requirement, designated $M \models \varphi$. In the case that the system does not satisfy the requirement, the model checker returns a counterexample, which is a system execution trace exemplifying the requirement violation. The complete work flow is shown in Fig. 1.

We use the symbolic model-checking tool: New e(X)tensible Model Verifier (nuXmv) [39] for our experiments because it is well-documented [40–42], freely available , and frequently used in industry [26, 31, 38, 43–52]. However,

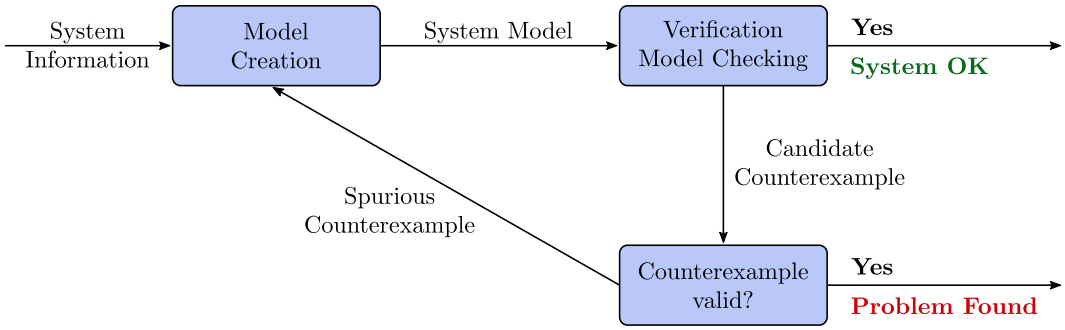



**Fig. 1** Model checking involves entering the system information (as a model) and requirements into a model-checking tool. If there is disagreement between the model's operation and its requirements, the model checker returns a *counterexample trace*. Otherwise, the system satisfies the specification.

there are several tools available with similar capabilities including Cadence's Symbolic Model Verifier (CadenceSMV) [53] (which has nearly identical syntax), Software Analysis Laboratory-Symbolic Model Checker (SAL-SMC) [54], Berkeley's ABC [55], and others []. nuXmv's expressible symbolic model verification (SMV) modeling language and the ease of integrating an add-on for contract-based design (Section III.C) sets it apart for our network design analysis.

### B. Linear Temporal Logic

Aerospace operational concepts are most frequently specified in terms of requirements over timelines [56–59], e.g., which represent possible scenarios. In our analysis, we choose a specification logic that naturally and intuitively encodes timelines: Linear Temporal Logic (LTL). LTL combines Boolean logic over system variables with how events unfold over time in different system scenarios. It is lightweight enough that model checkers can efficiently analyze requirements specified in LTL [38, 52, 60], yet we found it to be expressive enough to specify the pertinent requirements of our on-board network designs.

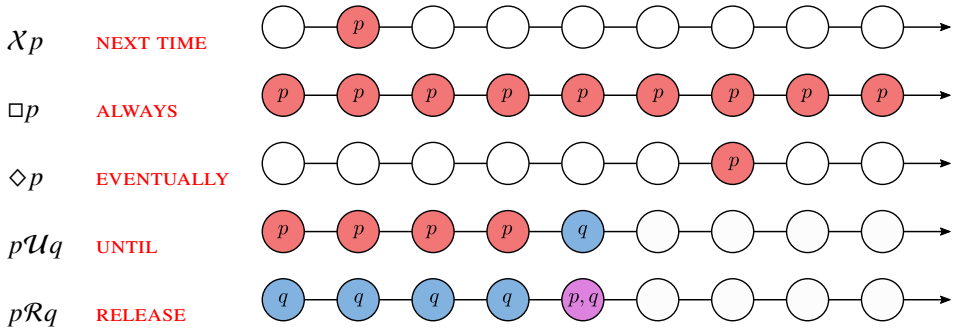
LTL reasons about time as a discrete series of timesteps, corresponding to the discrete nature of sensor readings, software variables, and packets sent over wireless networks. A trace through time can satisfy requirements over system variables as their values change at discrete timepoints. For the example Boolean variables $p$ and $q$, LTL can specify that $p$ must hold in the next time step after the current one ($\mathcal{X}p$); $p$ must always hold, i.e., remain invariant ($\Box p$); $p$ must be true eventually ($\Diamond p$); $p$ must hold until such a time when $q$ holds, where $q$ is guaranteed to hold eventually ($p\mathcal{U}q$); that an occurrence of $p$ releases $q$ from having to hold ($p\mathcal{R}q$); or any (possibly nested) combination of these. See Table 1 for an illustration.

**Linear Temporal Logic** (LTL) formulas reason about linear timelines; LTL requirements are formulas comprised of the following parts:

- finite set of atomic propositions {p, q}
- Boolean connectives: $\neg$, $\wedge$, $\vee$, and $\rightarrow$

- temporal connectives: $\mathcal{X}$, $\Box$, $\Diamond$, $\mathcal{U}$, and $\mathcal{R}$ (see Table 1)

**Table 1 Linear temporal logic connectives and satisfying traces**




For a thorough survey on Linear Temporal Logic Symbolic Model Checking, with examples in nuXmv's SMV modeling language, we refer the reader to [61].

### C. Contract-based Design

A *contract* is a clear and concise description of the expected behavior of a system. Contract-based design is an emerging paradigm for the design of complex, multi-component systems [42, 62–70]. We associate each component in the system with a contract. A contract is an LTL formula of the form Always($assumption \rightarrow guarantee$). Contracts implement *assume-guarantee reasoning*. In a contract, *assumptions* are the expectations the component has from the environment in which it operates, and *guarantees* are behavioral promises fulfilled by the component provided the environmental assumptions are met. Contract-based design allows for compositional modeling by breaking down a large system into smaller components. It facilitates efficient re-use of a component between several systems provided the contract holds in the changed environment, and simplifies cross-validation of different models or sub-models.

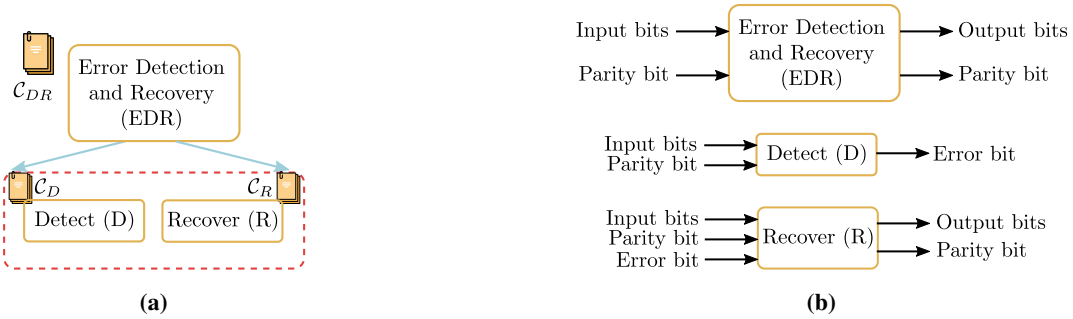



**Fig. 2** (a) Component hierarchy in a Error Detection and Recover (EDR) system composed from components Detect $D$ and Recover $R$. (b) Inputs and outputs for individual components.

Fig. 2a shows a simple network Error Detection and Recovery (EDR) system modeled compositionally. The system takes as input data bits, and a parity bit that is equal to the Exclusive-OR (XOR) of the data bits. The EDR system detects 1-bit errors (at most one data bit corrupted, parity bit intact), and uses the parity bit to recompute the faulty bit.

The system outputs the corrected data bits and the parity bit. The EDR system is composed of two subcomponents: Detect (D) and Recover (R). The former finds and outputs the bit number that is corrupted, whereas, the latter uses the parity bit to recover the corrupted bit and output the corrected data bits. Every (sub)component has an associated contract as follows:

1) **Error Detection,** $C_D$ - guarantees that a bit error is detected under the assumption that only one bit is corrupted in the input data.
2) **Error Recovery,** $C_R$ - guarantees that the corrupted data bit is recovered under the assumption that only one bit is corrupted in the input data.
3) **Error Detection and Recovery,** $C_{DR}$ - guarantees the input XOR parity equals the parity of the recovered bits under the assumption that only one bit is corrupted in the input data.

The system-level contract, $C_{DR}$, holds if and only if it is correctly refined by the contracts $C_D$ and $C_R$ of its subcomponents. A refinement is correct when the system-level contract's guarantee can be met by the guarantees of the subcomponents' contracts, and the guarantee of each (sub)component contract holds individually under the stated assumptions. A contract-based refinement analysis tool takes as input component descriptions and contracts, and automatically checks refinement. We use the Othello Contract Refinement and Analysis (OCRA) [63] tool for specifying contracts on the individual components of our wireless network, and checking refinement. Contract-based design provides extensibility because a library of components can be maintained and used in several designs; existing system subcomponents can be swapped with library components provided the contract refinement still holds.

**Contract-based Design with OCRA.** OCRA's main functionality is the verification of contract refinements, i.e., to check if a component satisfies its contract provided its subcomponents satisfy their respective contracts. OCRA takes as input a textual description of the component interfaces and their decomposition into subcomponents, each components' contracts, and their refinements with the contracts of the subcomponents. As an example, consider the Error Detection and Recovery (EDR) system of Fig. 2a that operates on an input data stream of two bits, and a parity bit. The OCRA specification of the system appears in Fig. 3. EDR is composed of two subcomponents: Detect D, and Recover R.

- EDR takes as input the corrupted data stream (`input`) and the XOR parity of the uncorrupted data stream (`parity`), and outputs the recovered uncorrupted data stream (`output`).
- D takes as input the corrupted data stream (`input`) and XOR parity (`parity`), and outputs the index of the corrupted bit in the data stream (`error_bit`).
- R takes as input the corrupted data stream (`input`), the XOR parity (`parity`) and the index of the corrupted bit (`error_bit`), and outputs the recovered uncorrupted data stream (`output`).

OCRA is successfully able to verify that the main contract `error_recovery` of EDR (Fig. 3c) is refined by the contracts `bit_0_error`, `bit_1_error`, `bit_0_recover` and `bit_1_recover` of the Detect D (Fig. 3a) and Recover R (Fig. 3b). For a thorough description of OCRA's input language and algorithms, we refer the reader to [71].

```
COMPONENT Detect
INTERFACE
  INPUT PORT input: array 0 .. 1 of boolean;
  INPUT PORT parity: boolean;
  OUTPUT PORT error_bit: 0 .. 1;

  CONTRACT bit_0_error
    assume: input[0] != parity xor input[1];
    guarantee: error_bit = 0;

  CONTRACT bit_1_error
    assume: input[1] != parity xor input[0];
    guarantee: error_bit = 1;
```

**(a)** Error Detection

```
COMPONENT Recover
INTERFACE
  INPUT PORT input: array 0 .. 1 of boolean;
  INPUT PORT parity: boolean;
  INPUT PORT error_bit: 0 .. 1;
  OUTPUT PORT output: array 0 .. 1 of boolean;

  CONTRACT bit_0_recover
    assume: error_bit = 0;
    guarantee: output[0] = parity xor input[1]
               and output[1] = input[1];

  CONTRACT bit_1_recover
    assume: error_bit = 1;
    guarantee: output[1] = parity xor input[0]
               and output[0] = input[0];
```

**(b)** Error Recovery

```
COMPONENT EDR system
INTERFACE
  INPUT PORT input: array 0 .. 1 of boolean;
  INPUT PORT parity: boolean;
  OUTPUT PORT output: array 0 .. 1 of boolean;

  CONTRACT error_recovery
    assume: parity != input[0] xor input[1];
    guarantee: parity = output[0] xor output[1];

  REFINEMENT
    SUB D: Detect;
    SUB R: Recover;

  CONNECTION D.input := input;
  CONNECTION D.parity := parity;
  CONNECTION R.input := input;
  CONNECTION R.parity := parity;
  CONNECTION R.error_bit := D.error_bit;
  CONNECTION output := R.output;

  CONTRACT error_recovery REFINEDBY
                  D.bit_0_error,
                  D.bit_1_error,
                  R.bit_0_recover,
                  R.bit_1_recover;
```

**(c)** Error Detection and Recovery

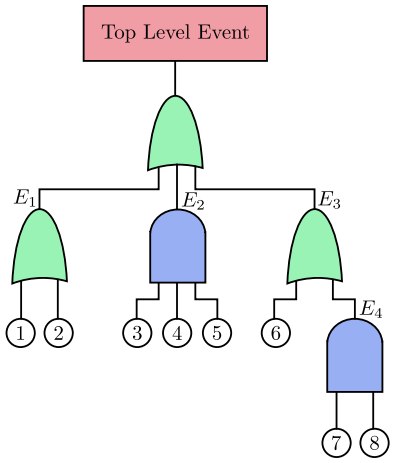
**Fig. 3** OCRA specification for a network Error Detection and Recovery (EDR) component made up of two subcomponents: Detect (D) and Recover (R). The contract `error_recovery` of the EDR component is refined by the contracts `bit_0_error`, `bit_1_error`, `bit_0_recover` and `bit_1_recover` of the Detect and Recover subcomponents.

It is important to note that when specifying the decomposition of EDR, we do not specify the actual behavior of Detect D and Recover R; we only specify the contracts and the component interface for performing refinement. It is up to the system designer to make sure that the implementations of D and R satisfy the components' contracts. OCRA allows behavioral specification of the leaf components (components that are not decomposed further in to subcomponents) as an SMV-language model (`ocra_print_system_implementation`), and allows checking if the components' contracts are satisfied by the model via model checking (`ocra_check_implementation`) with nuXmv. Non-leaf components do not have a corresponding SMV-language model, and their contracts are verified by composing the SMV models of their subcomponents. OCRA can automatically generate the SMV model for the complete system by recursive composition of the SMV models of subcomponents.

### D. Fault Tree Analysis

Fault-tree Analysis (FTA) is a top-down deductive failure analysis technique that uses Boolean logic to combine a series of undesired lower-level events and their combinations that lead to overall system failure. We use fault tree analysis to compare the robustness of different network models.

The results of fault analysis can be reported in several ways: as fault trees, in tables, or in failure propagation graphs.




**(a)** Combination of faults

| **Failure Probability** | **Logic Gate** | **Probability Equation** |
|---|---|---|
| P($E_1$) | OR | P(1) + P(2) |
| P($E_2$) | AND | P(3) × P(4) × P(5) |
| P($E_3$) | OR | P(6) + P($E_4$) |
| P($E_4$) | AND | P(7) × P(8) |
| P(TLE) | OR | P($E_1$) + P($E_2$) + P($E_3$) |

**(b)** Probability of failure

**Fig. 4** A sample fault-tree that uses Boolean logic to combine low level faults (in circles) that lead to an undesired state of a system, i.e. top-level event.

Fig. 4a shows a fault tree in which a boolean combination of lower-level events leads to an undesired state (top level event) of a system, i.e., a fault. The route through a tree between a fault and an initiating event is called a *Cut Set* and the shortest credible way through the tree from the fault to an initiating event is a *Minimal Cut Set*. The number of faults in a cut set represent it's *cardinality*. For example, in Fig. 4a the cut sets {1}, {7, 8}, and {3,4,5} are of cardinality 1, 2 and 3, respectively. Fault tree analysis helps in identifying cut sets of low cardinality, e.g., so that we can look for ways to remove them by adding fault tolerance to the system. We can compute the probability of occurrence of a top-level event by assigning probabilities to lower-level events (if they are known), or by characterizing the top-level event by a Boolean combination of its child events. For example, the probability of occurrence of event $E_1$ is the sum of probabilities of events 1 and 2 (Fig. 4b). We use the Extensible Safety Analysis Platform (xSAP) [72] tool that builds on nuXmv to perform fault-tree analysis of our wireless network and compute failure probabilities.

**Fault Tree Analysis with xSAP.** We refer to nuXmv models without faults as *nominal* models, and models with faults are called *extended* models. xSAP allows automatic and manual inclusion of faults to the nominal SMV-language models. We manually annotate our SMV models with three types of faults: Permanent, Transient, and Bounded. *Permanent* faults persist during the entire run of the system. *Transient* faults are erratic and may or may not occur during a system run. *Bounded* faults, as the name suggests, are time-bounded and resolve automatically after a finite number of time steps during a system run. For the Error Detection and Recovery (EDR) system of Fig. 3, a possible permanent fault occurs when the output bits of Recover R are always stuck at 0. We specify such faults by adding them to the SMV model of the component, and an external XML file (Fig. 5) specifies the failure probability and nominal value of each fault. For a thorough description of xSAP's input language and algorithms, see [73].

```
<?xml version="1.0"?>
<compass>
  <fmlist>
    <fm name="EDR.R.Bits_Stuck_At_Off" nominal_value="FALSE" probability="0.01"></fm>
    <fm name="EDR.D.Error_Stuck_At_One" nominal_value="FALSE" probability="0.02"></fm>
    <fm ... ></fm>
  </fmlist>
</compass>
```

**Fig. 5** Manual inclusion of faults to SMV model. The fault 'name' is included as a variable in the SMV model. The faulty behavior of the model is guarded by the fault variable.

### E. Fault-Tolerance in Data Systems

Reliability mechanisms for data and transmission have been extensively studied in the literature. We leverage techniques from experimental and theoretical results on storage systems [74–76], general system workloads [77–79], application-specific I/O patterns [80], and performance modeling [81]. Addressing reliability for data necessarily involves the inclusion of redundant information, either in the form of duplication of the data to be stored or transmitted, or by generating information that can be used to rebuild lost or damaged data in the case of a fault. System designers typically address reliability concerns in large-scale storage systems through the generation of *syndromes* that can be used to detect faults, prevent those faults from manifesting as failures, and repair those failures when new resources are available. The most basic type of syndrome that can be allocated is that of XOR parity [82]. Given a set of data to be stored or transmitted, we split this data into abstract sub-units of fixed or dynamic size called "blocks." We can treat each of these blocks as a vector of bytes and generate a syndrome by performing some calculation on each byte in the vector. In order to tolerate the loss of a single block in some set of $n$ blocks $B_0 \ldots B_{n-1}$ we must first compute a syndrome $P$ that allows for the recovery of any lost block within the set. One of the simplest methods for doing so is XOR parity:

$$P = B_0 \oplus B_1 \oplus B_2 \oplus \ldots \oplus B_{n-1}$$

This new block $P$ can then be stored and transmitted with the set, provided each block of the message is transmitted in such a way as to guarantee independence with respect to faults resulting in data loss. The loss of any one block, including the one containing $P$ will not result in the loss of data. If some block $B_j$ fails to transmit, or is corrupted, operations can still be performed on the set as normal by recomputing the value of the lost block. Given a loss of a block not containing $P$, we rebuild some lost block $B_j$ as

$$B_j = B_0 \oplus B_1 \oplus B_2 \oplus \ldots \oplus B_{j-1} \oplus B_{j+1} \oplus \ldots \oplus B_{n-1} \oplus P$$

(where $2 < j < n - 1$ for this example, but without loss of generality for other cases). If $P$ is lost, the data can be used normally, and $P$ can be recomputed as before.

Tolerating the loss of any two blocks requires calculating two independent syndromes; we refer to these as $P$ and $Q$,

which we calculate using $P$. We utilize the algebra of a Galois field $\mathbf{GF}(2^8)$ [83] for the computation of $Q$ to ensure the orthogonality of its construction. We utilize elements $g$ (generators of the Galois field) such that $g^n$ doesn't repeat until it has exhausted all elements of the field except $\{00\}$, where any numeral in $\{\}$ is a hexadecimally represented Galois field element. A full discussion of Galois field algebra is fully explored in the literature [84]. For $n$ blocks where $n \leq 255$ we compute:

$$\begin{aligned} P &= B_0 \oplus B_1 \oplus B_2 \oplus \ldots \oplus B_{n-1} \\ Q &= g^0 \cdot B_0 \oplus g^1 \cdot B_1 \oplus g^2 \cdot B_2 \oplus \ldots \oplus g^{n-1} \cdot B_{n-1} \end{aligned}$$

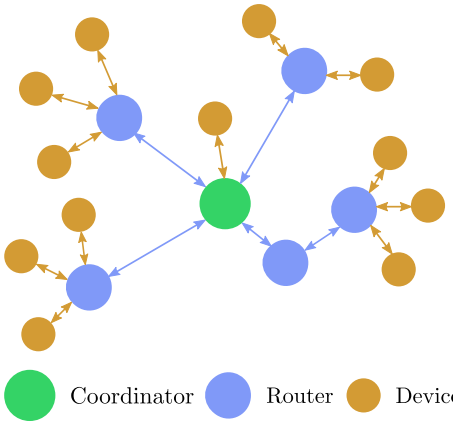
The loss of a single block can be recovered using the normal XOR parity method described in Section III.C. We can prevent the loss of $P$ or $Q$ since either can be recovered simply by recomputing using the above formulas. We can recover from the combined loss of any single non-parity block, and the loss of $Q$, by first recovering the non-parity block using XOR parity, and then recomputing $Q$. Recovering $P$, or the loss of two data blocks is somewhat more involved, and the discussion of the method is left to the literature [83].

### F. ZigBee Protocol

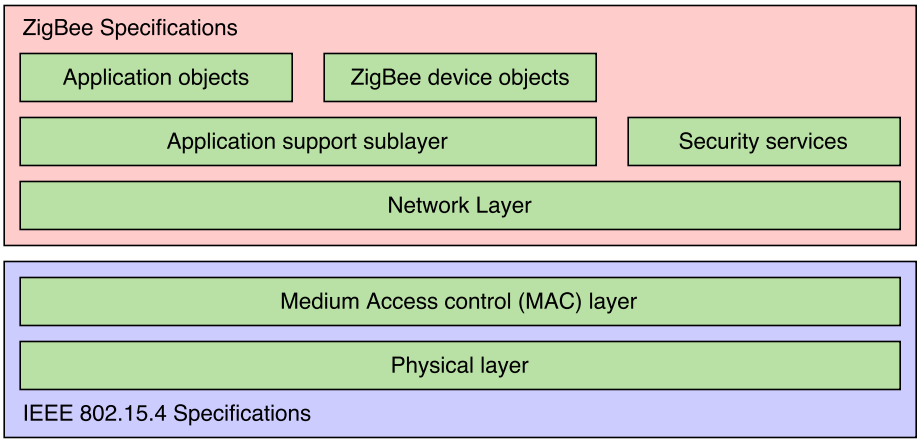
ZigBee is an IEEE 802.15.4 based specification protocol for small area networks. It is intended to be simpler and less expensive compared to Bluetooth or Wifi. Its transmission distance is up to 10-100 meters, dependent on power output and environmental characteristics. The transmitted data is secured by 128-bit symmetric encryption keys and has a maximum theoretical data rate of 250 kbits/sec. The ZigBee network layer natively supports star and tree networks, and generic mesh networking. Every ZigBee network must have one coordinator device, tasked with network creation, the control of its parameters and basic maintenance. The network is self-organized. It uses a node called the *coordinator* that acts as the primary ZigBee node and initiates network formation. The sensor nodes, or *end-devices* collect system values and send them to the coordinator. The sensor nodes can connect directly to the coordinator or through an intermediate *collector* or *router*. Fig. 6a shows communication between different nodes of a ZigBee network using a mesh topology. Coordinators and routers are always-on devices, whereas, end-devices look for an available coordinator when they power up. Fig. 6b shows an abstracted version of the ZigBee protocol stack split between the ZigBee specification and the IEEE 802.15.4 specification.

## IV. Trustworthy Messaging for Aviation Systems

Enabling communication between systems over wireless networks, especially in mission- and life-critical aviation systems, presents two primary challenges: privacy and fault-tolerance. Unlike wired systems, wireless systems are prone to spoofing and eavesdropping; it is critical that we are able to authenticate messages received wirelessly, and hide




**(a)**




**(b)**

**Fig. 6** (a) Example ZigBee network using a mesh topology showing communication between different nodes. (b) Layered architecture of the ZigBee 802.15.4 protocol stack

the contents of messages from those without the appropriate privileges. In addition, we must ensure messages can survive natural faults occurring within the system, and attempts by adversaries to induce faults or corrupt messages to prevent transmission. To address these challenges and build a trustworthy wireless messaging system for next-generation aviation systems we propose a novel, dynamic, messaging structure that combines fault-tolerance using parity, and Galois-field-based syndromes to improve fault-tolerance; PEKS-like [11, 16] regular expression based keywords and meta-keywords for privacy-preserving search [85, 86]; and a dynamically-sized payload as pictured in Fig. 7.

Information to be transmitted is first divided into packets as normal with the ZigBee protocol, but additional packets are inserted to form a set of packets that we refer to as a *burst*. Bursts are evaluated together for fault-tolerance reasons. Each burst of packets consists of a variable number of $N$ payload packets followed by two packets for fault-tolerance purposes computed using XOR- and Galois-field syndrome methods as discussed in Section III.E. The number of payload packets per burst can be configured at runtime and controlled adaptively to trade-off effective bandwidth for

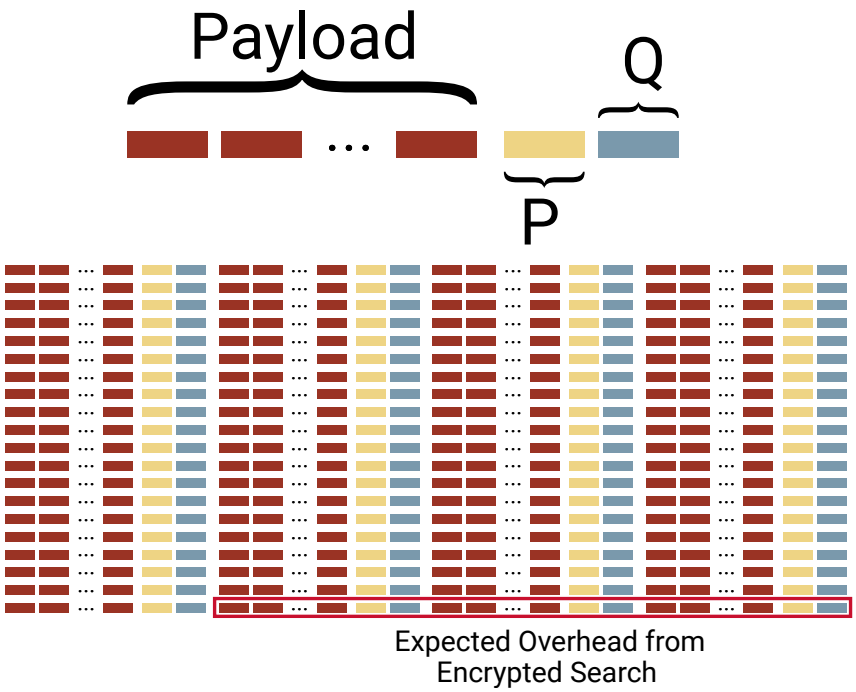



**Fig. 7** Anatomy of a file transmitted using secure, and reliable, packet bursts. Each burst consists of an adjustable number of payload packets carrying data or meta-data, and parity packets.

fault-tolerance and vice-versa. This gives our system the ability to alter fault-tolerance and throughput capabilities in response to changing conditions on the aircraft. When the network communication path becomes more congested, or in response to potential denial-of-service attacks, our system can drop the fault-tolerance of individual bursts to ensure packets can be transmitted effectively in the available bandwidth, and to ensure compliance with real-time deadlines. When bandwidth is plentiful, fault-tolerance can be improved. Fig. 8 shows the small modification necessary to enable burst-formation and interpretation with ZigBee Packets.

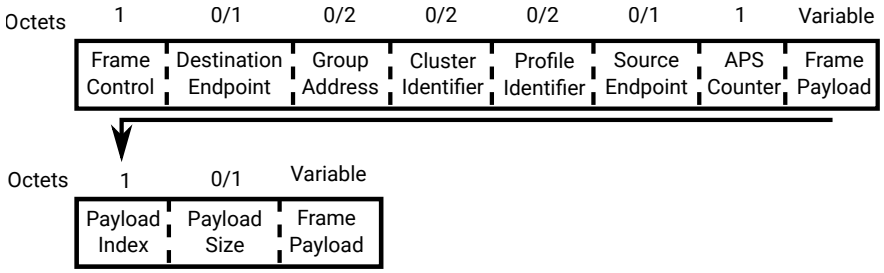



**Fig. 8** Modified format of ZigBee packet as part of a burst. Additional octets for payload 'index' and 'size' are added to store packet information for a burst.

Members of a burst allocate the first 1 or 2 octets of their packet frame's payload (an overhead of less than 1.5%) to represent the payload index (i.e. the packet's index within a burst) and in the case of the first packet in a payload, the

size of payload. Normal data follows. Since this new format adheres to existing ZigBee standards, it can be used with COTS equipment and requires only a software interpreter at the end points of routed messages to interpret. If packets are lost at any point during transmission, they can be rebuilt with high probability either by the ultimate receiver, or by intermediate nodes in the mesh network.

Each data object transmits with a further modification to add encryption using current standards for end-to-end encryption, such as ECCDHE [87, 88]. Trap-door encrypted keywords included with the transmission enable intermediaries to route and index information securely using the RESeED framework for regular-expression based search over encrypted data [85, 86]. This allows the data to be transmitted in a semi-oblivious manner. Individual nodes can route messages from any party along pathways without knowing their eventual destination, payload contents, or purpose. Authorized end-users and nodes can filter based on these trap-doors, or query nodes for available data without first decrypting the stored data. This prevents adversaries from determining the content of messages, their origination, and their purpose.

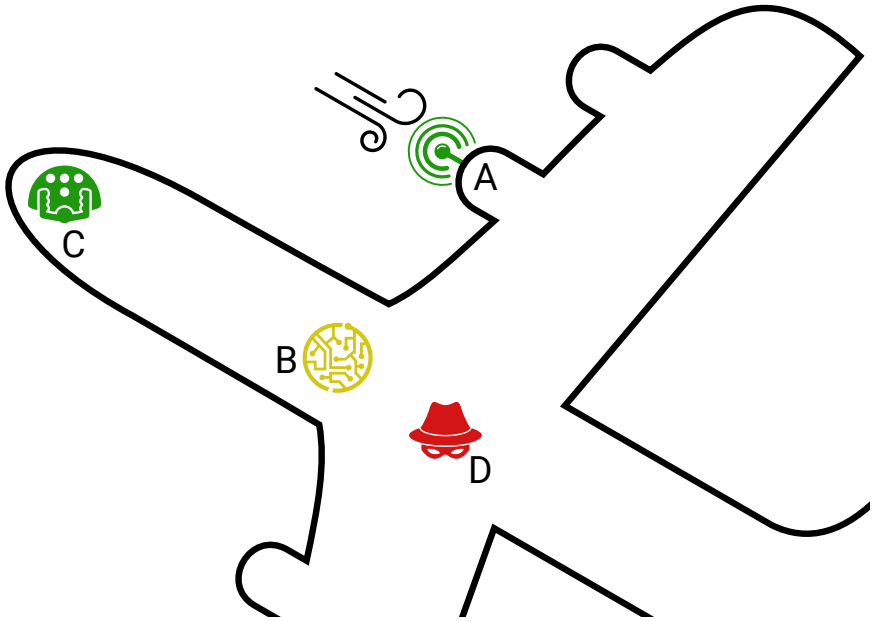



**Fig. 9** The pitot tube indicated by A wants to transmit information to the cockpit and autopilot indicated by C, but lacks the necessary wireless range. It must relay its message through the partially trusted intermediary, B, which is another subsystem of the aircraft.

Our proposed modifications to the ZigBee wireless standard provide additional trustworthy capabilities necessary for the specific challenges of next-generation aircraft communication network capabilities, such as those exemplified in Fig. 9. Given the large physical footprint of most aerospace systems, the ZigBee protocol cannot be implemented in a universal broadcast fashion in which every device can communicate directly with every other device in the large mesh network. Messages will instead have to be relayed, sometimes over multiple hops. It is even possible that every

intermediary that relays a message may not have the same permissions, or trust. Individual nodes may be sourced from different contractors, may have been patched for vulnerabilities at different dates, and may be subject to different threats, such as backdoors or physical compromise. In Fig. 9 a pitot tube (A) on the wing of the aircraft needs to communicate with the cockpit (C) to report observed airspeed. Because the cockpit is out of range for the transmitter (either permanently due to distance, or temporarily due to interference) it must relay its message through an unpatched intermediary (B) that represents another subsystem on the aircraft equipped with a ZigBee transmitter. An adversary (D) is attempting to eavesdrop on packets from (A), or potentially to forge messages from (A) through the compromised node (B). Because the sensor is encrypting all traffic with (C) through public/private key-pair encryption with searchability, the adversary does not succeed in accessing or altering the messages between (A) and (C).

The proposed modifications allow for recovery from faults without decrypting the underlying message, and allow for fault recovery either at intermediate nodes, or the eventual end-recipient of a message. Since the frame payload of each message is encrypted, adversaries cannot determine or modify the origination of packets or their contents. We can additionally employ Onion routing [89] as part of existing modifications of the ZigBee standard [90] to obscure message origination and destination [91].

## V. Formal Network Modeling

We present a formal fault analysis model of a basic ZigBee network. Network protocols are suitable candidates for contract-based verification since their layered architecture makes them amenable to compositional modeling. As the first step, we abstract the ZigBee protocol to a form suitable for model checking. The system model we use for fault analysis contains each component of the protocol stack of Fig. 6b except the *Security Services* subcomponents due to their added complexity; however, this does not change our proposed analysis procedure.

Our formal fault analysis methodology uses three tools: OCRA [63] for component-based modeling, contract-based design, and refinement; nuXmv [42] for model checking; and xSAP [72] for safety assessment and analysis. Fig. 10 shows the formal model of the wireless communication system to model data transfer between a ZigBee end-device node (Sensor) and ZigBee coordinator node (Cockpit) via a physical medium. The ZigBee coordinator and end-device are comprised of sub-models representing the protocol stack of Fig. 6b. Additional components, including the Protocol and Data Layers at the Sensor and Cockpit, model system behavior outside the control of the ZigBee device, which are often implemented as user software. We model data flow from the sensor to the cockpit, as shown in Fig. 11, however, other configurations are also possible using a similar modeling technique. We specify contracts for all components of Fig. 10. The leaf components have an additional behavioral specifications in the SMV language.

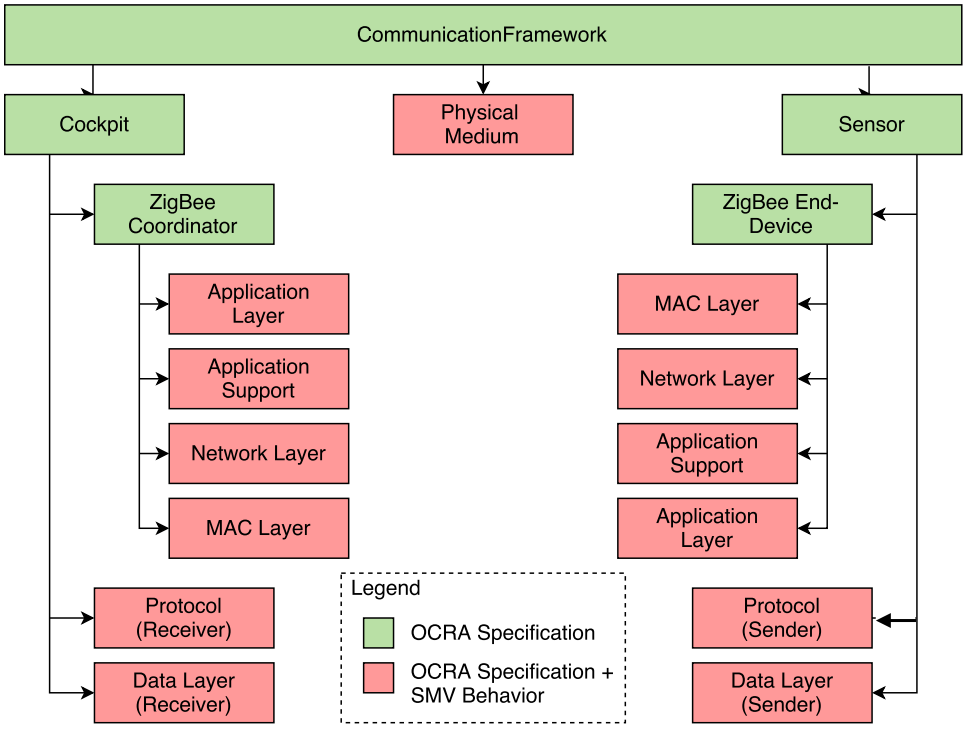



**Fig. 10** Formal model layout for the ZigBee wireless communication system

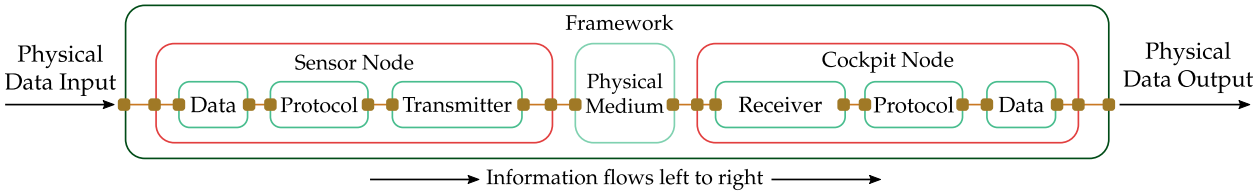



**Fig. 11** ZigBee wireless system as modeled in OCRA. The top-level system component is composed of three subcomponents: Sensor Node, Physical Medium, and Cockpit Node.

### A. Behavioral Specification

We only model the very high-level nominal behavior of the leaf components, i.e., the output of a leaf component equals its input. It is important to emphasize that this level of modeling is adequate for fault analysis since we are interested in inter-component communication failure, and not the correctness of the ZigBee protocol specification. Fig. 12a shows the nominal behavior specification of the `DataLayer` leaf component in the Sensor and Cockpit; specifications for other leaf components are similar. System behavior in the presence of faults, or extended behavior, is modeled by output of the component not being equal to the input. We include two types of faults: *permanent* and *transient* for which the dynamics are shown in Fig. 12b and 12c, respectively. The faults are derived from a subset of faults associated with a ZigBee network covered in existing literature [6, 92]. Table 2 shows the faults modeled in our extended wireless system. Fault Z1 models signal corruption due to electromagnetic interference in the communication

```
MODULE DataLayer
  IVAR
    input : boolean;
  DEFINE
    -- Output of component
    output := input;
```

**(a)** Nominal Behavior

```
MODULE PermanentFault
  VAR
    -- current fault state
    has_fault: boolean;
  IVAR
    -- controlled by xSAP
    fault_event: boolean;
  ASSIGN
    init(has_fault) := FALSE;
    next(has_fault) :=
          case
           (has_fault | fault_event): TRUE;
           TRUE: FALSE;
          esac;
```

**(b)** Permanent Fault Behavior

```
MODULE TransientFault
  VAR
    -- current fault state
    has_fault: boolean;
  IVAR
    -- controlledby xSAP
    fault_event: boolean;
  INIT
    has_fault = FALSE;
  TRANS
    next(has_fault) <-> fault_event;
```

**(c)** Transient Fault Behavior

```
MODULE DataLayer
  IVAR
    input: boolean;
  VAR
    -- Fault dynamics
    fault_S2__dynamic: PermanentFault;
  DEFINE
    -- Fault event (controlled by xSAP)
    fault_S2__event := fault_S2__dynamic.fault_event;
    -- Fault state
    fault_S2 := fault_S2__dynamic.has_fault;
    -- Output of component
    output:= fault_S2 ? input : !input;
```

**(d)** Off-Nominal Behavior

```
<?xml version="1.0"?>
<compass>
  <fmlist>
    <fm name="Sensor.DataLayer.fault_S2__event" nominal_value="FALSE" probability="0.01"></fm>
    <fm ... ></fm>
  </fmlist>
</compass>
```

**(e)** xSAP specification of modeled fault

**Fig. 12** SMV Behavior specification for nominal (a) and extended (d) behavior of components in the presence of Permanent (b) and Transient (c) faults. The fault are specified in a XML file (e).

spectrum. Faults Z2, Z3, and Z4 are general faults associated with a ZigBee network and model the end-device not being discoverable, coordinator not accepting new connections, and coordinator failing to set up the network, respectively. E1 models the failure of the error recovery mechanism, whereas S2 is a harder fault and models the failure of the sensor itself. Fig. 12d shows the extended behavior of the `DataLayer` component in the presence of fault S2. The fault event (`fault_S2__event`) in Fig. 12d is controlled by xSAP and specified in XML as shown in Fig. 12e. Specification for other components and fault combinations are modeled similarly.

### B. Contract Specification

We use OCRA to lay out the model of Fig. 10, and perform elementary refinement analysis; the contract of each component in this model is simply to output whatever appears at the input. Though our models do not take advantage of all of the capabilities of contract-based design, OCRA allows us to scalably manage the model as more components

**Table 2 Faults associated with wireless network analyzed in this work (S: sensor, R: cockpit)**

| Fault | Description | Mode | Authority |
|---|---|---|---|
| Z1 | Signal interference | Transient | Physical Medium |
| Z2 | End-Device not discoverable | Transient | Network Layer (S) |
| Z3 | Coordinator cannot accept new connections | Transient | Network Layer (R |
| Z4 | Coordinator fails to set up network | Permanent | Application Layer (R) |
| E1 | Error recovery mechanism fails | Transient | Protocol (S/R) |
| S2 | Sensor fails | Permanent | Data Layer (S) |

are added, and new behavior is specified. After we map the OCRA specification of each leaf component to the corresponding SMV behavior, we generate the composite SMV language specification of our model using the command (`ocra_print_system_implementation`).

## VI. Experimental Results

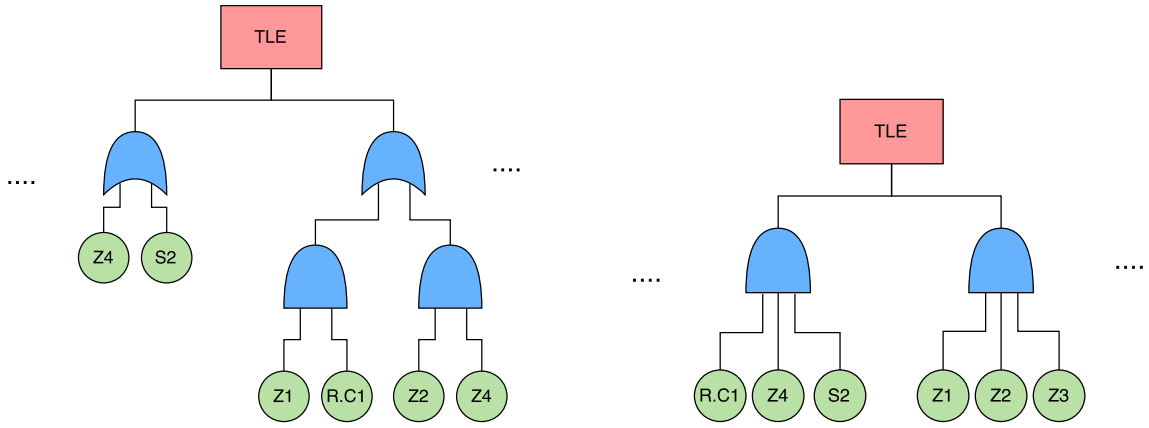
In order to assess fault tolerance, we model the occurrence of failures predicted by our fault tree constructions with the assumption that packet failures satisfy the Markovian assumptions, i.e., the process of packet failures is stateless and independent [93]. Under this assumption we evaluate our failures as a Poisson process [94] and solve numerically. We are primarily interested in evaluating the probability that a burst suffers 3 or more failures, as such situations are unrecoverable under our current error correction and fault-tolerance schema. This is primarily a function of the burst payload size, and we expect that as the number of payload packets increase per burst, fault-tolerance will drop, but bandwidth utilization will improve.

The probability that a burst will fail is given by

$$P(\text{fail}_{\text{burst}}) = 1 - \frac{\Gamma(\lfloor k+1 \rfloor, \lambda)}{\lfloor k \rfloor !}$$

where $\Gamma()$ is the incomplete-gamma function [95], $k$ is the maximum number of faults we can tolerate without burst failure, and $\lambda$ is the expected number of failures per burst. We work with $k = 2$ due to the limitations of our syndrome mechanisms. We find $\lambda$ as a function of the data-transmission rate $r$. This rate, $r$ is either 20kbps or 250kbps, depending on the band being used. We make the assumption that $r$ is 20kbps for the results presented in this paper, but we also computed the results for 250kbps with similar results. We represent the per-packet failure rate by $\mu$ and the configurable number of packets per burst by $b$. Lastly the size of the packet is $s$, which we assume to be 512 bits. With these substitutions our final expression for $\lambda$ is:

$$\lambda = \frac{\mu \cdot r}{s} \cdot \frac{b \cdot s}{r} = \mu b.$$




**(a)** Sample fault tree 1 **(b)** Sample fault tree 2

**Fig. 13** Fault tree using minimal cut sets for the extended ZigBee wireless system.

### A. Formal Fault Analysis

In order to generate estimates for the per-packet failure rate $\mu$, both for our experiment, and as a tool for system designers to deploy for runtime estimation during operation, we design models for the nominal and extended behavior of our ZigBee system. The top-level requirement for our nominal and extended system is "Data transmitted from the sensor is received by the cockpit." In LTL, one way to formally specify this requirement is:

$$\textsc{Always}(Sensor.DataLayer.input \rightarrow \quad \textsc{Eventually}(Cockpit.DataLayer.output \wedge (Cockpit.DataLayer.output = Sensor.DataLayer.input)))$$

For fault tree analysis, our top-level event is the negation of our top-level requirement. For the ZigBee wireless network, Fig. 13 shows fault trees for the extended model, and Fig. 14 and Fig. 15 show the cut sets and minimal cut sets generated by xSAP. After determining the points of failure, a failure function assigns probabilities to individual faults. The overall failure probability needs to be equal to or less than the failure probability of the wired system. Similarly, we can compare multiple wireless systems by generating fault trees from the same model with slight modifications, which shows the effect of model variations on robustness of the resulting system. After computing the fault trees, we associate probabilities with individual faults (these probabilities can be computed experimentally or can be derived from device data-sheets) to measure the failure probability of the top level event, i.e., the per-packet failure rate $\mu$. We may also choose to add fault-tolerance to the physical network to reduce the per-packet failure rate; we leave that extension to future work. The value of $\mu$ measures the performance of our modified ZigBee protocol in the empirical analysis.

$$
\begin{aligned}
CutSets &= (\{Z4, S2, Z1, R.E1, Z2\}, \{Z4, Z1, R.E1, Z2\}, \\
&\quad \{S2, Z1, R.E1, Z2\}, Z4, S2, \{Z1, R.E1\}, \{Z2, Z4\} \ldots) \\
MinCutSet &= (Z4, S2, \{Z1, R.E1\}, \{Z2, Z4\})
\end{aligned}
$$

**Fig. 14** Cut Sets and Minimal Cut Sets for fault tree in Fig. 13a.

$$
\begin{aligned}
CutSets &= (\{R.E1, Z4, S2, Z1, Z2, Z3\}, \{R.E1, Z4, S2\}, \\
&\quad \{Z1, Z2, Z3\} \ldots) \\
MinCutSet &= (\{R.E1, Z4, S2\}, \{Z1, Z2, Z3\})
\end{aligned}
$$

**Fig. 15** Cut Sets and Minimal Cut Sets for fault tree in Fig. 13b.

### B. Empirical Analysis

The empirical results of our analysis appear in Fig. 16 and 17 for a failure rate of $\mu = 0.005$ and $\mu = 0.1$, respectively. Note that $\mu$ can be determined using formal fault analysis of Section VI.A. Using our numerical model of fault tolerance, we plotted the expected percentage of untolerated burst failures occurring over any transmission epoch, and the percentage of bandwidth utilized for data transmission, accounting for the 4% overhead due to parity packets included in each burst, and the additional octets needed to encode the burst in the existing ZigBee Standard.

We evaluated our system for several different configurations in terms of payload packets per burst, and for two different failure rates of ZigBee packets. The literature [6] suggests that failure rates can vary from negligible rates (Fig. 16) to as high as 10% in extreme circumstances (Fig. 17). We present results where the failure rate per burst is no higher than roughly 0.03, representing an expected loss of at most 3% of bursts during transmission to show the impact on our tunable parameter (data packets per burst) on bandwidth utilization for data, and our ability to tolerate packet failures. As can be seen from the plots, our system is not only robust in the presence of failures, but the tunability allows systems to optimize performance with respect to the most important metric for a given situation, regardless of the current failure rate experienced by packets.

### C. Primary Takeaway

Our two main contributions: a framework for formal fault analysis, and additions to the ZigBee protocol to enhance reliability and trustworthiness of communication are intertwined. The former can be used to estimate packet failure rates of a new/existing wireless network (with or without fault tolerance mechanisms), while the latter can reduce failure rates by trading off reliability and available bandwidth. Compared to other techniques for fault analysis, the presented formal model is more scalable; new components or component behavior can be specified in the SMV language and contract-refinement can be used to ensure nothing breaks. Our analysis is also exhaustive: the formal model allows for

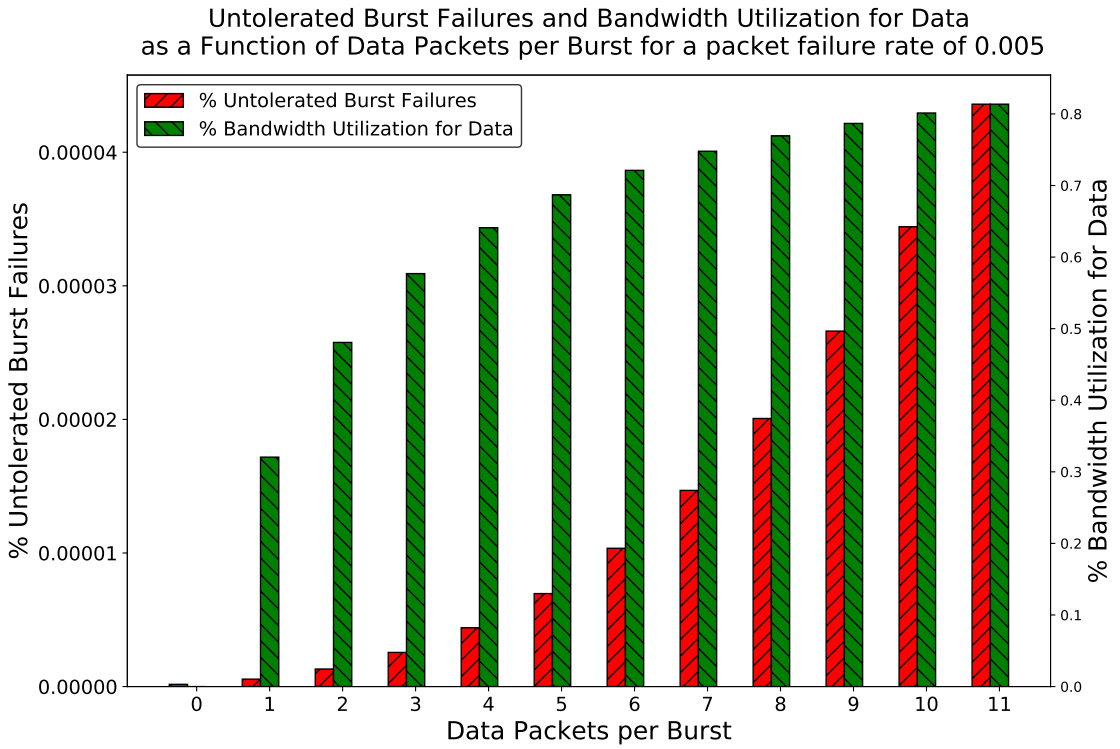



**Fig. 16** The impact of per-packet failure rate $\mu = 0.005$ on untolerated failures, and the percentage of bandwidth utilized for productive data transfer for per-packet failure rates.

exhaustive exploration of the behavior space of the modeled network. The modeled network can then be physically implemented using off-the-shelf components, while employing the proposed ZigBee modifications, and tuning to optimize performance.

## VII. Conclusions and Future Work

We demonstrate a proof of concept for the development of a formal framework for fault analysis of wireless communication networks. The formal model is plug-and-play in the sense that new wireless protocol behavior models can be plugged in easily in to the composite system model with minor modifications.

Future extensions of the work includes quantitative assessment of failure probabilities, introducing more behavior and fault extensions to the models, and removing some obvious faults. A thorough analysis of fault tolerant architectures by adding extensions for redundant end-devices (sensors), coordinators (cockpit devices) and communication medium (wires, different frequencies) will help designers in identifying optimum fault tolerance techniques. Extending the technique to include more wireless communication protocols will help better identify wired components of the aircraft than can be migrated to wireless. A desirable extension will be automatic introduction of fault tolerant architectures to achieve a desired failure rate probability.

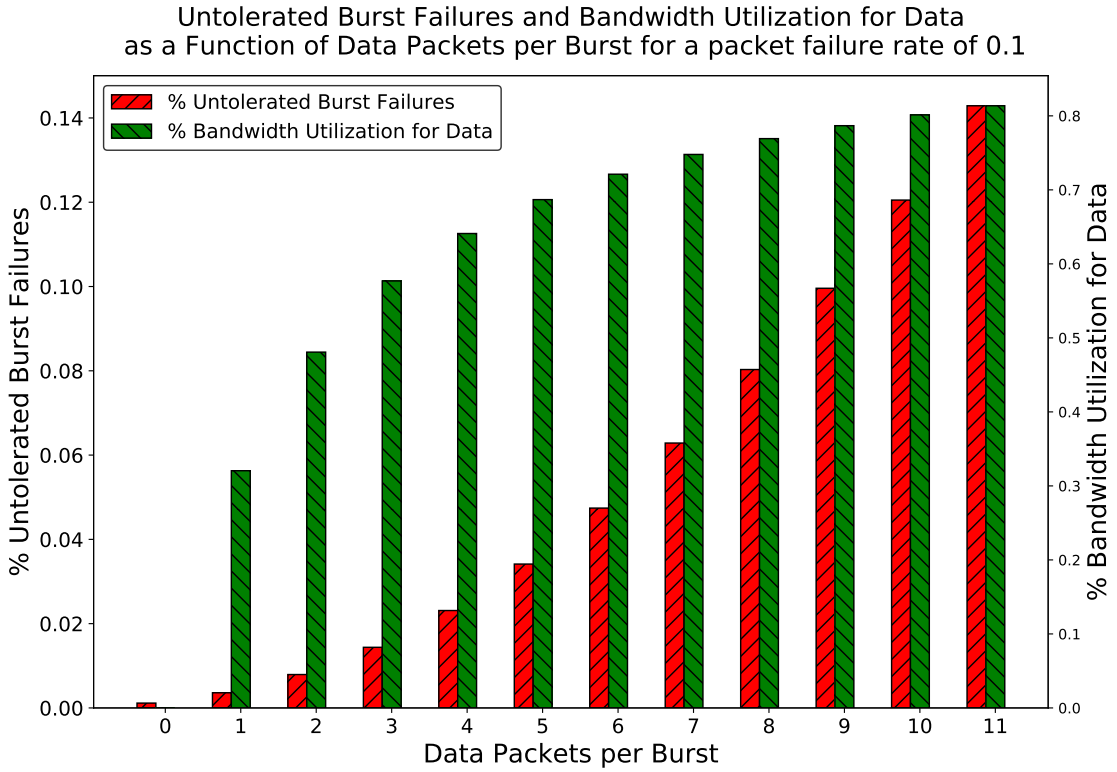



**Fig. 17** The impact of per-packet failure rate $\mu = 0.1$ on untolerated failures, and the percentage of bandwidth utilized for productive data transfer for per-packet failure rates.

We intend to implement our proposed extension to the ZigBee framework for dynamic fault-tolerance and bandwidth utilization to show its efficacy in practice, COTS deployability, and to demonstrate how simple automated reasoning on-board an aircraft could be used to estimate the current packet failure rate and adaptively modify the burst configuration to meet mission-critical goals for transmission.

## Acknowledgments

We thanks the anonymous reviewers for their valuable feedback and comments. Thanks to NSF CAREER Award CNS-1664356 and NASA's Efficient Reconfigurable Cockpit Design and Fleet Operations using Software Intensive, Networked and Wireless Enabled Architecture (ECON) Grant NNX15AQ84G for supporting this work.

## References

[1] Canaday, H., "War on Wiring," *Aerospace America*, 2017, pp. 24–27. URL `https://aerospaceamerica.aiaa.org/features/war-on-wiring/`.

[2] Dureja, R., Rozier, E. W., and Rozier, K. Y., "A Case Study in Safety, Security, and Availability of Wireless-Enabled Aircraft Communication Networks," *17th AIAA Aviation Technology, Integration, and Operations Conference*, AIAA, 2017.

https://doi.org/10.2514/6.2017-3112.

[3] Attia, S. B., Cunha, A., Koubâa, A., and Alves, M., "Fault-tolerance mechanisms for ZigBee wireless sensor networks," *Work-in-Progress (WiP) session of the 19th Euromicro Conference on Real-Time Systems (ECRTS 2007), Pisa, Italy*, 2007, pp. 37–40.

[4] Wan, J., Wu, J., and Xu, X., "A novel fault detection and recovery mechanism for zigbee sensor networks," *Future Generation Communication and Networking (FGCN). Second International Conference on*, Vol. 1, IEEE, 2008, pp. 270–274. https://doi.org/10.1109/FGCN.2008.105.

[5] Alena, R., Ellis, S. R., Hieronymus, J., and Maclise, D., "Wireless Avionics and Human Interfaces for Inflatable Spacecraft," *Aerospace Conference*, IEEE, 2008, pp. 1–16. https://doi.org/10.1109/AERO.2008.4526527.

[6] Alena, R., Gilstrap, R., Baldwin, J., Stone, T., and Wilson, P., "Fault tolerance in ZigBee wireless sensor networks," *Aerospace Conference*, IEEE, 2011, pp. 1–15. https://doi.org/10.1109/AERO.2011.5747474.

[7] Ulusar, U. D., Celik, G., Turk, E., Al-Turjman, F., and Guvenc, H., "Practical Performability Assessment for ZigBee-Based Sensors in the IoT Era," 2019, pp. 21–31. https://doi.org/10.1007/978-3-319-93557-7_2.

[8] Song, D. X., Wagner, D., and Perrig, A., "Practical techniques for searches on encrypted data," *Security and Privacy (S&P), IEEE Symposium on*, IEEE Comput. Soc, 2000, pp. 44–55. https://doi.org/10.1109/secpri.2000.848445, URL https://doi.org/10.1109%2Fsecpri.2000.848445.

[9] Goh, E.-J., "Secure Indexes," , 2003. URL http://eprint.iacr.org/, report 2003/216.

[10] Curtmola, R., Garay, J., Kamara, S., and Ostrovsky, R., "Searchable symmetric encryption: Improved definitions and efficient constructions," IOS Press, 2011, pp. 895–934. https://doi.org/10.3233/jcs-2011-0426.

[11] Boneh, D., and Waters, B., "Conjunctive, subset, and range queries on encrypted data," *Theory of cryptography*, Springer, 2007, pp. 535–554. https://doi.org/10.1007/978-3-540-70936-7_29.

[12] Wang, C., Ren, K., Yu, S., and Urs, K. M. R., "Achieving usable and privacy-assured similarity search over outsourced cloud data," *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 2012, pp. 451–459. https://doi.org/10.1109/infcom.2012.6195784.

[13] Ibrahim, A., Jin, H., Yassin, A. A., Zou, D., and Xu, P., "Towards Efficient Yet Privacy-Preserving Approximate Search in Cloud Computing," *The Computer Journal*, Vol. 57, No. 2, 2013, pp. 241–254. https://doi.org/10.1093/comjnl/bxt045, URL https://doi.org/10.1093%2Fcomjnl%2Fbxt045.

[14] Kamara, S., and Lauter, K., "Cryptographic cloud storage," *Financial Cryptography and Data Security*, Springer, 2010, pp. 136–149. https://doi.org/10.1007/978-3-642-14992-4_13.

[15] Hofheinz, D., and Weinreb, E., "Searchable encryption with decryption in the standard model." *IACR Cryptology ePrint Archive*, Vol. 2008, 2008, p. 423. https://eprint.iacr.org/2008/423.

[16] Boneh, D., Di Crescenzo, G., Ostrovsky, R., and Persiano, G., "Public key encryption with keyword search," *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2004, pp. 506–522. https://doi.org/10.1007/978-3-540-24676-3_30.

[17] Zhao, Y., and Rozier, K. Y., "Formal Specification and Verification of a Coordination Protocol for an Automated Air Traffic Control System," *Proceedings of the 12th International Workshop on Automated Verification of Critical Systems (AVoCS 2012)*, Electronic Communications of the EASST, Vol. 53, European Association of Software Science and Technology, 2012.

[18] Zhao, Y., and Rozier, K. Y., "Formal Specification and Verification of a Coordination Protocol for an Automated Air Traffic Control System," *Science of Computer Programming Journal*, Vol. 96, No. 3, 2014, pp. 337–353. https://doi.org/10.1016/j.scico.2014.04.002.

[19] Zhao, Y., and Rozier, K. Y., "Probabilistic Model Checking for Comparative Analysis of Automated Air Traffic Control Systems," *Proceedings of the 33rd IEEE/ACM International Conference On Computer-Aided Design (ICCAD 2014)*, IEEE/ACM, San Jose, California, U.S.A., 2014, pp. 690–695. https://doi.org/10.1109/iccad.2014.7001427.

[20] Mattarei, C., Cimatti, A., Gario, M., Tonetta, S., and Rozier, K. Y., "Comparing Different Functional Allocations in Automated Air Traffic Control Design," *Proceedings of Formal Methods in Computer-Aided Design (FMCAD 2015)*, IEEE/ACM, Austin, Texas, U.S.A, 2015. https://doi.org/10.1109/fmcad.2015.7542260.

[21] Gario, M., Cimatti, A., Mattarei, C., Tonetta, S., and Rozier, K. Y., "Model Checking at Scale: Automated Air Traffic Control Design Space Exploration," *Proceedings of 28th International Conference on Computer Aided Verification (CAV 2016)*, LNCS, Vol. 9780, Springer, Toronto, ON, Canada, 2016, pp. 3–22. https://doi.org/10.1007/978-3-319-41540-6_1.

[22] Groce, A., Havelund, K., Holzmann, G., Joshi, R., and Xu, R.-G., "Establishing flight software reliability: testing, model checking, constraint-solving, monitoring and learning," *Annals of Mathematics and Artificial Intelligence*, Vol. 70, No. 4, 2014, pp. 315–349. https://doi.org/10.1007/s10472-014-9408-8.

[23] Mehlitz, P. C., "Trust your model-verifying aerospace system models with Java pathfinder," *Aerospace Conference, 2008 IEEE*, IEEE, 2008, pp. 1–11. https://doi.org/10.1109/aero.2008.4526573.

[24] Can, A. B., Bultan, T., Lindvall, M., Lux, B., and Topp, S., "Eliminating synchronization faults in air traffic control software via design for verification with concurrency controllers," *Automated Software Engineering*, Vol. 14, No. 2, 2007, pp. 129–178. https://doi.org/10.1007/s10515-007-0008-2.

[25] Munoz, C., Carreño, V., and Dowek, G., "Formal analysis of the operational concept for the small aircraft transportation system," *Rigorous Development of Complex Fault-Tolerant Systems*, Springer, 2006, pp. 306–325. https://doi.org/10.1007/11916246_16.

[26] Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V. Y., Noll, T., and Roveri, M., "The COMPASS approach: Correctness, modelling and performability of aerospace systems," *Computer Safety, Reliability, and Security*, Springer Berlin Heidelberg, 2009, pp. 173–186. https://doi.org/10.1007/978-3-642-04468-7_15.

[27] Chan, W., Anderson, R. J., Beame, P., Burns, S., Modugno, F., Notkin, D., and Reese, J. D., "Model checking large software specifications," *Software Engineering, IEEE Transactions on*, Vol. 24, No. 7, 1998, pp. 498–520. https://doi.org/10.1109/32.708566.

[28] Sreemani, T., and Atlee, J. M., "Feasibility of model checking software requirements: A case study," *Proceedings of 11th Annual Conference on Computer Assurance (COMPASS)*, IEEE, 1996, pp. 77–88. https://doi.org/10.1109/cmpass.1996.507877.

[29] von Essen, C., and Giannakopoulou, D., "Analyzing the next generation airborne collision avoidance system," *Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2014, pp. 620–635. https://doi.org/10.1007/978-3-642-54862-8_54.

[30] Bozzano, M., Bruintjes, H., Cimatti, A., Katoen, J.-P., Noll, T., and Tonetta, S., "COMPASS 3.0," *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, edited by T. Vojnar and L. Zhang, Springer International Publishing, Cham, 2019, pp. 379–385. https://doi.org/10.1007/978-3-030-17462-0_25.

[31] Bozzano, M., Cimatti, A., Pires, A. F., Jones, D., Kimberly, G., Petri, T., Robinson, R., and Tonetta, S., "Formal Design and Safety Analysis of AIR6110 Wheel Brake System," *Proceedings of the 27th International Conference on Computer Aided Verification (CAV)*, Springer, 2015, pp. 518–535. https://doi.org/10.1007/978-3-319-21690-4_36.

[32] Radio Technical Commission for Aeronautics (RTCA), "DO-178B: Software Considerations in Airborne Systems and Equipment Certification," , December 1992.

[33] Radio Technical Commission for Aeronautics, "DO-178C/ED-12C – Software Considerations in Airborne Systems and Equipment Certification," , 2012. URL https://www.rtca.org/content/standards-guidance-materials.

[34] Radio Technical Commission for Aeronautics, "DO-333 – Formal Methods Supplement to DO-178C and DO-278A," , 2011. URL https://www.rtca.org/content/standards-guidance-materials.

[35] Radio Technical Commission for Aeronautics (RTCA), "DO-254: Design Assurance Guidance for Airborne Electronic Hardware," , April 2000.

[36] Clarke, E. M., Emerson, E. A., and Sistla, A. P., "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems*, Vol. 8, No. 2, 1986, pp. 244–263. https://doi.org/10.1145/5397.5399.

[37] Queille, J. P., and Sifakis, J., "Specification and verification of concurrent systems in CESAR," *International Symposium on Programming*, LNCS, Vol. 137, Springer Berlin Heidelberg, 1982, pp. 337–351. https://doi.org/10.1007/3-540-11494-7_22.

[38] Dureja, R., and Rozier, K. Y., "More Scalable LTL Model Checking via Discovering Design-Space Dependencies ($D^3$)," *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, edited by D. Beyer and M. Huisman, Springer International Publishing, Cham, 2018, pp. 309–327. https://doi.org/10.1007/978-3-319-89960-2_17.

[39] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., and Tonetta, S., "The nuXmv Symbolic Model Checker," *CAV*, Springer International Publishing, 2014, pp. 334–342. https://doi.org/10.1007/978-3-319-08867-9_22.

[40] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M., "NuSMV: A New Symbolic Model Checker," *International Journal of Software Tools for Technology Transfer (STTT)*, Vol. 2, No. 4, 2000, pp. 410–425. https://doi.org/10.1007/s100090050046.

[41] R. Cavada, A. C., Jochim, C., Keighren, G., Olivetti, E., Pistore, M., Roveri, M., and Tchaltsev, A., "NuSMV 2.4 User Manual," Tech. rep., CMU/ITC-irst, 2005.

[42] Bozzano, M., Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., and Tonetta, S., "nuXmv 2.0 User Manual," Tech. rep., FBK - Via Sommarive 18, 38055 Povo (Trento) – Italy, 2019.

[43] Raimondi, F., Lomuscio, A., and Sergot, M. J., "Towards Model Checking Interpreted Systems," *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, ACM Press, 2002, pp. 115–125. https://doi.org/10.1145/860575.860792.

[44] Gribaudo, M., Horváth, A., Bobbio, A., Tronci, E., Ciancamerla, E., and Minichino, M., "Model-Checking Based on Fluid Petri Nets for the Temperature Control System of the ICARO Co-generative Plant," Tech. rep., Berlin, Heidelberg, 2002. https://doi.org/10.1007/3-540-45732-1_27.

[45] Tribble, A., and Miller, S., "Software Safety Analysis of a Flight Management System Vertical Navigation Function-A Status Report," *22nd Digital Avionics Systems Conference Proceedings (DASC)*, IEEE, 2003, pp. 1.B.1–1.1–9 v1. https://doi.org/10.1109/dasc.2003.1245805.

[46] Choi, Y., and Heimdahl, M., "Model Checking Software Requirement Specifications Using Domain Reduction Abstraction," *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, IEEE Comput. Soc, 2003, pp. 314–317. https://doi.org/10.1109/ase.2003.1240328.

[47] Miller, S. P., Tribble, A. C., Whalen, M. W., and Heimdahl, M. P. E., "Proving the shalls," *International Journal on Software Tools for Technology Transfer*, Vol. 8, No. 4-5, 2006, pp. 303–319. https://doi.org/10.1007/s10009-004-0173-6.

[48] Miller, S. P., "Will This Be Formal?" *Theorem Proving in Higher Order Logics*, edited by O. A. Mohamed, C. Muñoz, and S. Tahar, Springer, Berlin, Heidelberg, 2008, pp. 6–11. https://doi.org/10.1007/978-3-540-71067-7_2.

[49] Yoo, J., Jee, E., and Cha, S., "Formal Modeling and Verification of Safety-Critical Software," *Software, IEEE*, Vol. 26, No. 3, 2009, pp. 42–49. https://doi.org/10.1109/ms.2009.67.

[50] Gan, X., Dubrovin, J., and Heljanko, K., "A Symbolic Model Checking Approach to Verifying Satellite Onboard Software," *Science of Computer Programming (2013)*, Vol. 82, 2014, pp. 44–55. https://doi.org/10.1016/j.scico.2013.03.005.

[51] Lahtinen, J., Valkonen, J., Björkman, K., Frits, J., Niemelä, I., and Heljanko, K., "Model checking of safety-critical software in the nuclear engineering domain," *Reliability Engineering & System Safety*, Vol. 105, No. 0, 2012, pp. 104 – 113. https://doi.org/10.1016/j.ress.2012.03.021.

[52] Dureja, R., and Rozier, K. Y., "FuseIC3: An algorithm for checking large design spaces," *Formal Methods in Computer Aided Design (FMCAD)*, 2017, pp. 164–171. https://doi.org/10.23919/FMCAD.2017.8102255.

[53] McMillan, K., "The SMV Language," Tech. rep., Cadence Berkeley Labs, 1999.

[54] de Moura, L., Owre, S., Rueß, H., Rushby, J., Shankar, N., Sorea, M., and Tiwari, A., "SAL 2," *Computer Aided Verification*, edited by R. Alur and D. A. Peled, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 496–500. https://doi.org/10.1007/978-3-540-27813-9_45.

[55] Brayton, R., and Mishchenko, A., "ABC: An Academic Industrial-Strength Verification Tool," *Computer Aided Verification*, edited by T. Touili, B. Cook, and P. Jackson, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 24–40. https://doi.org/10.1007/978-3-642-14295-6_5.

[56] Chang, R., et al., "SatCarrier CONOPS: A Concept of Operations for Satellite Carriers," Tech. rep., Citeseer, 2009.

[57] Erzberger, H., and Heere, K., "Algorithm and operational concept for resolving short-range conflicts," *Proc. IMechE G J. Aerosp. Eng.*, Vol. 224, No. 2, 2010, pp. 225–243. https://doi.org/10.1243/09544100JAERO546.

[58] Ryan, J., Cummings, M., Roy, N., Banerjee, A., and Schulte, A., "Designing an interactive local and global decision support system for aircraft carrier deck scheduling," *Infotech@ Aerospace 2011*, ????, p. 1516.

[59] Mindock, J., Reilly, J., Urbina, M., Rubin, D., Hailey, M., Reyes, D., Hanson, A., Burba, T., Cerro, J., McGuire, K., et al., "Exploration Medical Capability ConOps and Systems Engineering Technical Interchange Meeting Summary," 2017.

[60] Clarke, E. M., Henzinger, T. A., Veith, H., and Bloem, R., *Handbook of model checking*, Springer, 2018. https://doi.org/10.1007/978-3-319-10575-8.

[61] Rozier, K. Y., "Linear Temporal Logic Symbolic Model Checking," *Computuer Science Review*, Vol. 5, No. 2, 2011, p. 163–203. https://doi.org/10.1016/j.cosrev.2010.06.002.

[62] Cimatti, A., and Tonetta, S., "A property-based proof system for contract-based design," *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, IEEE, 2012, pp. 21–28. https://doi.org/10.1109/seaa.2012.68.

[63] Cimatti, A., Dorigatti, M., and Tonetta, S., "OCRA: A tool for checking the refinement of temporal contracts," *Automated Software Engineering (ASE)*, IEEE, 2013, pp. 702–705. https://doi.org/10.1109/ase.2013.6693137.

[64] Cimatti, A., and Tonetta, S., "Contracts-refinement proof system for component-based embedded systems," *Science of Computer Programming*, Vol. 97, 2015, pp. 333–348. https://doi.org/10.1016/j.scico.2014.06.011.

[65] Gómez-Martínez, E., Rodríguez, R. J., Elorza, L. E., Rezabal, M. I., and Earle, C. B., "Model-based verification of safety contracts," *International Conference on Software Engineering and Formal Methods*, edited by C. Canal and A. Idani, Springer International Publishing, Cham, 2015, pp. 101–115. https://doi.org/10.1007/978-3-319-15201-1_7.

[66] Cimatti, A., and Tonetta, S., "A Temporal Logics Approach to Contract-Based Design," *Architecture-Centric Virtual Integration (ACVI)*, IEEE, 2016, pp. 1–3. https://doi.org/10.1109/acvi.2016.7.

[67] Quinton, S., and Graf, S., "Contract-based verification of hierarchical systems of components," *Software Engineering and Formal Methods (SEFM)*, IEEE, 2008, pp. 377–381. https://doi.org/10.1109/sefm.2008.28.

[68] Graf, S., Passerone, R., and Quinton, S., "Contract-Based Reasoning for Component Systems with Rich Interactions," *Embedded Systems Development*, Springer New York, 2013, pp. 139–154. https://doi.org/10.1007/978-1-4614-3879-3_8.

[69] Nuzzo, P., Sangiovanni-Vincentelli, A. L., Bresolin, D., Geretti, L., and Villa, T., "A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems," *Proceedings of the IEEE*, Vol. 103, No. 11, 2015, pp. 2104–2132. https://doi.org/10.1109/JPROC.2015.2453253.

[70] Bozzano, M., Cimatti, A., Mattarei, C., and Tonetta, S., "Formal Safety Assessment via Contract-Based Design," *Automated Technology for Verification and Analysis*, Springer International Publishing, 2014, pp. 81–97. https://doi.org/10.1007/978-3-319-11936-6_7.

[71] Cimatti, A., Dorigatti, M., and Tonetta, S., "OCRA: Othello Contracts Refinement Analysis User Guide (Version 1.3)," , ???? URL https://es-static.fbk.eu/tools/ocra/download/OCRA_Language_User_Guide.pdf.

[72] Bittner, B., Bozzano, M., Cavada, R., Cimatti, A., Gario, M., Griggio, A., Mattarei, C., Micheli, A., and Zampedri, G., "The xSAP Safety Analysis Platform," *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Springer Berlin Heidelberg, 2016, pp. 533–539. https://doi.org/10.1007/978-3-662-49674-9_31.

[73] "xSAP: The eXtended Safety Assessment Platform User Guide (Version 1.2)," , ???? URL https://es.fbk.eu/tools/xsap/downloads/xsap-manual.pdf.

[74] Schroeder, B., and Gibson, G. A., "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, USENIX Association, USA, 2007, p. 1–es. https://doi.org/10.5555/1267903.1267904.

[75] Rozier, E., Belluomini, W., Deenadhayalan, V., Hafner, J., Rao, K., and Zhou, P., "Evaluating the impact of undetected disk errors in raid systems," *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, IEEE, 2009, pp. 83–92. https://doi.org/10.1109/dsn.2009.5270353.

[76] Hafner, J. L., Deenadhayalan, V., Belluomini, W., and Rao, K., "Undetected disk errors in RAID arrays," *IBM Journal of Research and Development*, Vol. 52, No. 4.5, 2008, pp. 413–425. https://doi.org/10.1147/rd.524.0413.

[77] Wallace, G., Douglis, F., Qian, H., Shilane, P., Smaldone, S., Chamness, M., and Hsu, W., "Characteristics of Backup Workloads in Production Systems," *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, USENIX Association, USA, 2012, p. 4. https://doi.org/10.5555/2208461.2208465.

[78] Tarasov, V., Kumar, S., Ma, J., Hildebrand, D., Povzner, A., Kuenning, G., and Zadok, E., "Extracting Flexible, Replayable Models from Large Block Traces," 2012, p. 22. https://doi.org/10.5555/2208461.2208483.

[79] Anderson, E., "Capture, Conversion, and Analysis of an Intense NFS Workload," *Proccedings of the 7th Conference on File and Storage Technologies*, USENIX Association, USA, 2009, p. 139–152. https://doi.org/10.5555/1525908.1525919.

[80] Madhyastha, H. V., McCullough, J. C., Porter, G., Kapoor, R., Savage, S., Snoeren, A. C., and Vahdat, A., "Scc: Cluster Storage Provisioning Informed by Application Characteristics and SLAs," *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, USENIX Association, USA, 2012, p. 23. https://doi.org/10.5555/2208461.2208484.

[81] Soundararajan, G., Lupei, D., Ghanbari, S., Popescu, A. D., Chen, J., and Amza, C., "Dynamic Resource Allocation for Database Servers Running on Virtual Storage," *Proccedings of the 7th Conference on File and Storage Technologies*, USENIX Association, USA, 2009, p. 71–84. https://doi.org/10.5555/1525908.1525914.

[82] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A., "RAID: high-performance, reliable secondary storage," *ACM Computing Surveys*, Vol. 26, No. 2, 1994, pp. 145–185. https://doi.org/10.1145/176979.176981.

[83] Anvin, H. P., "The mathematics of RAID-6," , 2007.

[84] Jacobson, N., *Lectures in Abstract Algebra: III. Theory of Fields and Galois Theory*, Vol. 32, Springer Science & Business Media, 2012. https://doi.org/10.1007/978-1-4612-9872-4.

[85] Salehi, M. A., Caldwell, T., Fernandez, A., Mickiewicz, E., Rozier, E. W., Zonouz, S., and Redberg, D., "RESeED: Regular expression search over encrypted data in the cloud," *Cloud Computing (CLOUD), IEEE 7th International Conference on*, IEEE, 2014, pp. 673–680. https://doi.org/10.1109/cloud.2014.95.

[86] Salehi, M. A., Caldwell, T., Fernandez, A., Mickiewicz, E., Rozier, E. W. D., Zonouz, S., and Redberg, D., "RESeED: A secure regular-expression search tool for storage clouds," *Software: Practice and Experience*, Vol. 47, No. 9, 2017, pp. 1221–1241. https://doi.org/10.1002/spe.2473.

[87] Joux, A., "A One Round Protocol for Tripartite Diffie?Hellman," Springer Science and Business Media LLC, 2004. https://doi.org/10.1007/s00145-004-0312-y.

[88] Bernstein, D. J., "Curve25519: New Diffie-Hellman Speed Records," *Public Key Cryptography (PKC)*, Springer Berlin Heidelberg, 2006, pp. 207–228. https://doi.org/10.1007/11745853_14.

[89] Goldschlag, D., Reed, M., and Syverson, P., "Onion routing," *Communications of the ACM*, Vol. 42, No. 2, 1999, pp. 39–41. https://doi.org/10.1145/293411.293443.

[90] Nezhad, A. A., Miri, A., and Makrakis, D., "Location privacy and anonymity preserving routing for wireless sensor networks," *Computer Networks*, Vol. 52, No. 18, 2008, pp. 3433–3452. https://doi.org/10.1016/j.comnet.2008.09.005.

[91] Chakrabarty, S., John, M., and Engels, D. W., "Black routing and node obscuring in IoT," *Internet of Things (WF-IoT), IEEE 3rd World Forum on*, IEEE, 2016, pp. 323–328. https://doi.org/10.1109/wf-iot.2016.7845477.

[92] Singh, A., , and Snigdh, I., "Modelling Failure Conditions in Zigbee based Wireless Sensor Networks," *International Journal of Wireless and Microwave Technologies*, Vol. 7, No. 2, 2017, pp. 25–34. https://doi.org/10.5815/ijwmt.2017.02.03.

[93] Markov, A. A., *The theory of algorithms*, Vol. 15, 1960.

[94] Law, A. M., and Kelton, D. M., *Simulation Modeling and Analysis*, 3rd ed., McGraw-Hill Higher Education, 1999. https://doi.org/10.5555/554952.

[95] Abramowitz, M., and Stegun, I. A., "Handbook of mathematical functions," *Applied mathematics series*, Vol. 55, No. 62, 1966, p. 39.