Book chapter for
"Computational Models, Software Engineering and Advanced Technologies in Air Transportation"

Chapter Title: A Distributed Systems Approach to Airborne Self-Separation

Authors: Henrique Moniz (1), Alessandra Tedeschi (2), Nuno Ferreira Neves (1), Miguel Correia (1)

(1) University of Lisboa, Portugal
(2) Deep Blue s.r.l., Rome, Italy

Corresponding Author: Henrique Moniz (hmoniz@di.fc.ul.pt)

Abstract

This chapter introduces the reader to the benefits of distributed computing in air transportation. It presents a solution to airborne self-separation based on RAPTOR, a stack of distributed protocols that allows aircraft to reach different types of agreement in the presence of faults, both of accidental and malicious nature. These protocols are used as primitives to implement specific services for airborne self-separation, which are created within the context of a conflict resolution algorithm based on game theory.

1 INTRODUCTION

Air Traffic Management (ATM) is concerned with the management of air traffic flow in a safe and efficient manner. In its current form, ATM is based on rigid off-line flight planning and ground-based air traffic controllers (ATCOs). The airspace is divided into sectors, and each sector is assigned an ATCO team, which becomes its central authority. The ACTOs are responsible for maintaining horizontal and vertical separation among aircraft, while ensuring an orderly and expeditious air traffic flow. This task is performed by issuing directions to aircraft and by providing flight context information to pilots, such as routes to waypoints and weather conditions.

The current controller-based approach to ATM relies heavily on controllers' skills, with little or no autonomy for pilots and companies. Moreover, it does not scale up to cope with the increasing volume of future air traffic, which is expected to grow at a rate of 5 to 6 percent per year (Eurocontrol, 1999). Several alternative solutions and complementary approaches to overcome the limits of current ATM are actively under investigation. Recent advances in technologies are making possible a new concept of ATM, namely *airborne self-separation* (FAA/Eurocontrol, 2001). It represents a concept in which the responsibility for aircraft separation is shifted from the ground to the air and where pilots are allowed to select their flight paths without any external intervention by air traffic controllers (Nordwall, 1995; Perry, 1997). In a future self-separation environment, pilots will have more responsibility for the safe and efficient conduction of the flight and they should be supported by an automated decision-support system that processes all available information, thus assisting the pilot in optimizing the aircraft trajectory while maintaining separation among aircraft.

In general terms, the economic advantages of airborne self-separation will manifest in two ways. First, it should lead to reduced costs. Self-optimization by the airlines could be more effective than any global optimization that can be performed by a human controller (RTCA Task Force 3, 1995). This is because different airlines might give higher priority to different parameters. One airline might prefer to optimize fuel consumption, while other might prefer to optimize flight delays. These preferences depend on company strategy or other factors only known to the airline and crew. Second, the global expansion in capacity of air traffic volume, due to the departure from the centralized and human-centric approach, will allow airlines to meet an increasing demand for air transportation.

This new approach, however, will not emerge without its share of technical issues. As a major paradigm shift, the deployment of airborne self-separation will also unveil a whole new domain of threats to the safety of air transportation. The consequent automation brought in by self-separation unveils a class of attacks that target these navigational systems in increasingly inconspicuous ways. Aircraft will need to rely on information provided by other aircraft in their vicinity to ensure proper coordination. Thus, it becomes imperative that this information is exchanged in a reliable and secure manner. Unfortunately, wireless communication is inherently unreliable, and the presence of a single malicious aircraft with the ability to transmit incoherent information or jam communication brings unpredictable and potentially catastrophic consequences for the safety of aircraft. Techniques must be adopted that allow aircraft to coordinate their maneuvering even if communication among them is subject failures, regardless of their nature - accidental or malicious.

This chapter presents an approach to airborne self-separation, taken from the discipline of distributed computing, that systematically addresses the problem of fault-tolerant decentralized coordination. In a distributed system, at the core of any kind of coordinated activity lies the need for some sort of agreement among the processes that compose the system (Guerraoui & Schiper, 1997; Turek & Shasha, 1992). Reaching agreement in the presence of faults is a fundamental and non-trivial problem in the distributed systems literature - a topic subject of countless papers. See (Cristian, 1991; Correia, Verissimo, & Neves, 2006; Fischer, 1983) for surveys. Within the context of this work, the distributed system represents the airborne self-separation environment, and the processes correspond to the aircraft.

The main thrust of this chapter is divided in two parts. The first part describes Satisficing Game Theory (SGT), introduced in (Johnson, Hill, Archibald, Frost, & Stirling, 2005), which is an approach for conflict resolution that can be applied to airborne self-separation where agents (i.e., aircraft) collaboratively exchange information to reach a solution for the group. SGT is offered as a case study that is representative of the limitations commonly present in current approaches, which do not take into account the potential hostility of the environment. In particular, this part identifies common gaps between the assumptions and the environment that lead to the need for mechanisms to provide consistent information in the presence of faults.

The second part of our contribution introduces one of such mechanisms. It presents a set of agreement protocols, organized into a stack called RAPTOR, which allows processes of a distributed system to reach different types of agreement. These agreement protocols are particularly adapted to wireless communications and operate correctly even if message transmissions are unreliable and some aircraft purposely exchange wrongful or inconsistent information. The protocols are designed within the context of a system model that appropriately captures the airborne self-separation environment. A rigorous correctness proof is also provided with each protocol to show that the properties of the protocol are met for the given model.

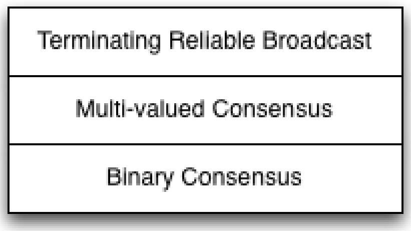

Figure 1. RAPTOR protocol stack.

The agreement protocols, depicted in Figure 1, are *binary consensus*, *multi-valued consensus*, and *terminating reliable broadcast*. Binary consensus is the most basic protocol and it allows processes to agree on a common binary value. This form of agreement is mostly used as a building block to construct other agreement protocols, but it can also be used to implement useful services. Imagine that a group of aircraft needs to perform some sort of maneuver where either all do the maneuver or none does. They can execute an instance of the binary consensus protocol. If the result is 1, then they all commit to the maneuver. If the result is 0, then none commits to the maneuver.

Above binary consensus is multi-valued consensus. This protocol uses binary consensus as a primitive and lets processes reach agreement on a value from an arbitrary domain. One example of the usefulness of this protocol is shown by the rank consistency service, described later in this chapter. The service resorts to multi-valued consensus to ensure that aircraft agree on a common ranking of the aircraft within a group. This ranking then determines how aircraft maneuver in order to avoid collisions.

Finally, terminating reliable broadcast allows a process to broadcast a message, giving both sender and receivers the guarantee that the message is uniformly delivered at all processes. For example, this protocol can be employed by an aircraft that wishes to disseminate some information to every aircraft within a certain radius. The view augmentation service, also described later in the chapter, implements a similar functionality.

As hinted above, the protocols of RAPTOR are used as primitives to implement specific services for airborne self-separation, namely *group membership*, *rank consistency*, and *view augmentation*. These services ensure that any complex coordination algorithm (e.g., SGT) is executed upon information consistent amongst all aircraft, which leads to a deterministic and safe maneuvering in the airspace.

Additionally, since the RAPTOR protocols can tolerate both accidental and malicious faults, the correct operation of the services is guaranteed even in adverse conditions.

The remainder of the chapter is organized as follows. Section 2 provides some background. The SGT approach to airborne-self separation is analyzed in Section 3. The RAPTOR protocols and related services are presented in Section 4. The future trends are discussed in Section 5. Finally, Section 6 concludes the chapter.

2 BACKGROUND

In the past decades, several feasibility studies about the airborne self-separation operational concept have been carried out (FAA/Eurocontrol, 2001; Wing, 2005). This research led to the proposal of a variety of interesting and effective techniques for the detection and resolution of conflicts in a self-separation environment. Most of them are described in two, very exhaustive, surveys (Kuchar & Yang, 2000; Dimarogonas & Kyriakopoulos, 2002). No approach, however, is found in the literature that applies the principles of fault-tolerant distributed systems to airborne self-separation.

The protocols of RAPTOR descend from a lineage of randomized consensus protocols that were proposed in 1983 by Ben-Or and Rabin (Ben-Or, 1983; Rabin, 1983). The consensus problem is the most typical form of agreement in distributed systems. Basically it can be described as follows. Each process starts with a private value that is used as a proposal value for consensus, and then, when a conclusion is reached, all processes obtain the same decision, usually on one of the proposed values. Behind this simple definition, consensus and other agreement problems in general end up being fairly complex, particularly when one starts to consider potential system errors. In fact, consensus has been proven to be deterministically impossible to solve in certain types of systems. The FLP impossibility result, named after its authors' last names, states that consensus is impossible to solve in asynchronous systems (i.e., systems that make no assumptions on the processes' computation and communication delays) if only one process can fail by crashing (Fischer, Lynch, & Paterson, 1985). A similar result exists for synchronous systems (i.e., systems with known bounds for the processes' computation and communication delays) if the transmissions associated with just one process can fail (Santoro & Widmeyer, 1989). The aforementioned protocols of Ben-Or and Rabin, however, circumvent the FLP impossibility result. Instead of trying to solve consensus deterministically, which is impossible, they utilize a technique called *randomization* to reach agreement in a probabilistic way.

The RAPTOR protocols are bound by the impossibility result of (Santoro & Widmeyer, 1989) for synchronous systems with transmission faults. The result is circumvented also by employing randomization at the level of the binary consensus protocol. This is the only protocol that actually needs to be randomized because all other protocols are built on top of this one. Previous to RAPTOR, some of the authors designed a stack of agreement protocols named RITAS, but this stack was designed for asynchronous systems and not with wireless communications in mind (Correia, Neves, & Verissimo, 2006; Moniz, Neves, Correia, & Veríssimo, 2006). RITAS was evaluated in a car-to-car communication scenario and, although the results were feasible for that particular application, it was clear that the protocols were not specially fit for wireless communications (Moniz, Neves, Casimiro, & Veríssimo, 2007). This was due to the fact that RITAS relies on a reliable point-to-point communication model, which forces the implementation of end-to-end message delivery mechanisms (e.g., TCP) that significantly increase the medium access contention.

Fault-tolerant agreement protocols for wireless networks represent a relatively recent research within the scientific community. Chockler et al. propose consensus algorithms for systems where nodes fail only by crashing and messages are lost due to collisions (Chockler, Demirbas, Gilbert, Newport, & Nolte, 2005). They resort to a specialized failure detector, which determines when message collisions occur and allows nodes to take some recovery actions. Koo et al. focus on a weaker problem, reliable broadcast, in multi-hop radio networks where each node adheres to a pre-determined transmission schedule (Koo, Bhandari, Katz, & Vaidya, 2006). Their algorithm tolerates faults of malicious nature, however, it also relies on collision-detection information to solve the problem and assumes that messages are only lost due to collisions. Finally, Drabkin et al. present a reliable broadcast protocol for wireless ad-hoc networks in asynchronous systems (Drabkin, Friedman, & Segal, 2005). Their protocol extends the system model with three types of failure detectors (mute, verbose, and trust) that detect messages sent too often, messages not sent, and incorrect nodes. The broadcast protocol is then built by combining together these failure detectors along with the use of digital signatures and gossiping.

3 CASE STUDY: SATISFICING GAME THEORY

A particular approach for airborne self-separation is presented as a case study in order to exemplify the limitations of current solutions and how they can be dealt with the protocols of RAPTOR. The approach resides on a decision-making algorithm that aims to provide adequate separation between aircraft by deciding how aircraft should maneuver to avoid potential conflicts. It is based on the framework of game theory for multi-agent systems where there is no centralized control and there is a clear global objective function that needs to be optimized.

Usually, the main difficulty in applying game theory to multi-agent systems is that it is hard to define the global objective function in terms of what is best for each agent. SGT tries to overcome such limit by seeking an adequate solution for the multi-agent system, rather than the optimal solution (Stirling, 2005). Basically, SGT defines the objective function by means of two utility functions, selectability and rejectability, which represent benefits (selectability) and costs (rejectability) for the choices of each agent. There may exist dependence between utility functions of different agents. In order to specify and analyze such system, utility functions can be conveniently viewed as marginal of a multivariate global probability function, in which dependences are expressed as conditional probabilities between variables. This way, a directed graph can be used to specify the system, and a solution can be obtained through standard Bayesian analysis.

In airborne self-separation, SGT can be applied as described in (Johnson, Hill, Archibald, Frost, & Stirling, 2005). Selectability and rejectability of each aircraft represent the benefits and costs of the aircraft's maneuvering choices. Benefits are essentially proportional to the optimality of the possible route. Costs, on the other hand, are proportional to the risk of collision with another aircraft. Correctly defining dependence between utility functions is the most important design factor because dependencies that are not necessary lead to unsatisfactory solutions. Thus, first of all, influence flows between aircraft are created. Influence flows are basically oriented graphs where nodes represent aircraft and edges represent relation between directional choices of aircraft. In order to create influence flows, at each time step, every aircraft exchanges information with all other aircrafts within a certain radius. This information is then used to rank the aircraft according to a certain policy. The information may include current position, destination, actual heading, and all the characteristics of the flight that can be considered useful for the ranking of the aircraft and the creation of influence flows. Then, the selectability of any particular aircraft is conditioned upon the selectability of all higher-ranked aircraft that are within a certain proximity range. Rejectability of agents is unconditioned: each aircraft responds to threats with exclusive self-interest. SGT scales well to high number of aircraft, since two important simplification can be safely applied: indirect influences between selectability functions can be discarded, and each aircraft can represent all viewable aircraft as a single entity that summarizes their maneuvering choices.

3.1 SGT LIMITATIONS

SGT has several shortcomings when applied to real-world scenarios. It makes two implicit assumptions about the information used to compute aircraft maneuvering: (1) information is complete and homogeneous, and (2) information is fresh. Homogeneous means that aircraft have no contradictory or incomplete information about any other aircraft. Fresh means that it always reflects the current state of other aircraft. For the SGT algorithm to function under these assumptions with a mere information exchange (i.e., every aircraft broadcasts its individual information), the system must be synchronous (i.e., information exchange can happen only at fixed time steps) and communication links must be reliable (i.e., no message can be lost at each time step). While the synchrony assumption is reasonable since on-board GPS units can provide very accurate clock synchronization, the reliability assumption is not.

Regrettably, in a real environment, the wireless communication medium is inherently unreliable. For instance, if messages are transmitted within overlapping time intervals, then a message collision occurs leading to message losses. This may create failure scenarios that can disrupt SGT operation. For instance, if two aircraft in conflicting courses have inconsistent or outdated information about each other, it is possible for each of them to derive maneuvering decisions that may further put them into a conflict. Additionally, if one of the communication subsystems fails, even if temporarily, then an incident can happen because both aircraft might be convinced that it is the responsibility of the other aircraft to maneuver around. There is also the threat of malicious activity, where aircraft can send wrong or incomplete information or no information at all. This can be performed in a calculated way as to maximize the chances of forcing an incident between two aircraft.

These failures can occur in the real world, but even if rare, they can lead to catastrophic consequences. Thus, it becomes imperative that they are dealt with in some way. The following section presents a set of protocols to support the exchange of information in a consistent manner, even in the presence of unreliable communications and malicious attacks.

4 RAPTOR

This section describes RAPTOR, a stack of agreement protocols particularly fit for wireless communication environments such as airborne self-separation. RAPTOR represents a bridge between the limitations identified in SGT and the actual conditions of the airborne self-separation environment. The protocols are distributed algorithms that are concurrently run by a set of n processes (i.e., aircraft). A number of message exchanges are performed as part of the execution of the algorithms. Each process executes an instance of a particular algorithm, and each instance outputs the same value, allowing them to reach various sorts of agreement. The protocols are also resilient to arbitrary communication failures (i.e., the communication links of the processes are allowed to fail). The only restriction is that no more than f processes may simultaneously experience failures in their outgoing links, with f being no more than a third of the total number of processes. In other words, the total number of processes must be $n \geq 3f+1$.

Section 4.1 describes the distributed system model, containing the assumptions that properly model the airborne self-separation environment.

The protocols are then described in their own respective sections. Section 4.2, 4.3, and 4.4 present binary consensus, multi-valued consensus, and terminating reliable broadcast, respectively. These sections adhere to the same structure. Each one starts with an informal discussion of the protocol, followed by the formal properties and a step-by-step description of the protocol. The protocol is illustrated in an accompanying figure. Next, it is outlined a hypothetical run of the protocol to help the reader better understand how it works in practice. Finally, each section ends with a rigorous mathematical proof showing that the algorithm satisfies the formal properties of the protocol. Each property is addressed individually and some lemmas are constructed and proved in order to support other proofs.

Section 4.5 wraps up this section. It bridges the theoretical protocols with specific services for SGT and airborne self-separation in general – group membership, rank consistency, and view augmentation. It informally discusses how easily the protocols of RAPTOR can be applied to implement these services.

4.1 SYSTEM MODEL

The airborne self-separation environment is modeled as a set of processes (i.e., the aircraft) that exchange information through wireless communication. The system is composed by a static set[1] of *n* processes $\prod = \{p_1, p_2, \ldots p_n\}$ that exchange messages through two communication primitives: *broadcast* and *receive*. The primitive broadcast(t,$\prod$,m) transmits a message *m* to all processes in set $\prod$ at step *t*. The primitive receive(t,$\prod$) returns a set with the messages that arrived due to invocations of *broadcast* from the processes in $\prod$ at step *t*. In some circumstances, the receive primitive can also be invoked as receive(t,S), where *S* is an arbitrary subset of $\prod$. In this case, the process only receives the messages broadcasted by the processes in *S*.

The system is synchronous, meaning that both the processing times of processes and the communications delays are bound by known constants. Communication between processes occurs at synchronous steps. At each step *t*, every process $p_i \in \prod$ executes the following actions: (1) invokes broadcast(t,$\prod$,m), (2) invokes receive(t,$\prod$), and (3) performs a state transition based on its current state and the set of values in the received messages. These values are stored by each process p_i in a local set V_i. Additionally, the protocols make extensive use of a function $\#_v(V_i)$ that returns the number of occurrences of value *v* in the set V_i.

The failure model follows closely the *communication failure model* introduced in (Santoro & Widmeyer, 1989), in which the communication links are subject to transmission failures. A message transmission is defined as a pair (α, β), where α is the message value sent by the source process, if any, and β is the value received by the destination process, if any. A transmission of a message amongst a source process p_i and a destination process p_j is faulty if one of the following occurs:

- Omission. The message sent by p_i is not received by p_j.
- Addition. A message is received by p_j when no such message was sent.
- Corruption. The message sent by p_i is received by p_j with different content.

A broadcast by a source process p_i originates *n* transmissions, one for each process in $\prod$, and faults can non-uniformly occur at any of these transmissions. The system can exhibit any type of transmission fault with the restriction that faults cannot affect more than *f* source processes per step, where $n \geq 3f + 1$ is the total number of processes. A fault is said to affect a source process p_i if it occurs at a transmission made by p_i.

The particularity of this model is that processes are modeled as not to exhibit faulty behavior, i.e., they correctly follow the protocols until termination. The notion of a faulty process is instead captured by the assumption of faulty message transmissions. This is because the effect of a crashed or malicious process is always reflected in message omissions or corruptions. For example, a process p_i whose communication systems are malfunctioning is captured by omission faults originating at p_i. Similarly, a malicious process p_i that sends messages with different content to every other

[1] The assumption of a fixed group of processes does not model the environment in a complete way. In reality, the groups are dynamic due to the aircraft being moving constantly. The fixed group assumption is present for the sake of simplicity because it allows a much clearer understanding of the protocols. Nevertheless, at Section 4.5.1 we informally describe a protocol that allows the creation of dynamic groups and, consequently, the practical applicability of the protocols.

process is captured by corruption faults originating at p_i. In order to properly capture the malicious behavior of processes into the model, it is also assumed that integrity-checking mechanisms cannot detect this kind of message corruption. Otherwise, corruption faults could be easily converted into omission faults.

Finally, each process has access to a local random bit generator through a function coin_flip() that returns unbiased bits. This is required since one of the protocols (i.e., binary consensus) requires processes to propose random values in certain situations.

4.2 BINARY CONSENSUS

The binary consensus represents the most basic form of agreement (i.e., on a single bit of information) and it is used as a building block for the other protocols provided by RAPTOR. In the protocol, each process p_i proposes a bit value x_i and all processes decide on the same result $d \in \{0,1\}$. Additionally, if all processes propose the same initial value *v*, then the decision has necessarily to be *v*.

4.2.1 Properties

Formally, the properties of the protocol are defined as follows:

- **BC1 (Validity).** If all processes propose the same value *v*, then any process that decides, decides *v*.
- **BC2 (Agreement).** No two processes decide differently.
- **BC3 (Termination).** Every process decides within *r* rounds with a probability *P* that approaches 1 with the number of rounds: $\lim_{r \to \infty} P = 1$.

The termination property (BC3) is formulated in a probabilistic way. This is done by defining a probability *P* for the processes to reach a decision that approaches 1 as the number of rounds increases (consensus algorithms are usually round-based). This basically states, in probabilistic terms, that the processes eventually reach a decision. The probabilistic definition of the termination property is required because there is an impossibility result for the communication failure model, which states that any form of deterministic agreement is unattainable if there can be $n-1$ or more transmission failures at any particular step. Adjusting the termination property in probabilistic terms allows this result to be circumvented using, for instance, *randomization* techniques. This approach has been used in the past to circumvent a similar impossibility result (Fischer, Lynch, & Paterson, 1985) with great success in terms of real world performance of the algorithms (Moniz, Neves, Correia, & Veríssimo, 2006). This happens because the fault pattern required to significantly delay the execution of the algorithm is very improbable to happen in practice and difficult to force by a malicious adversary.

Algorithm 1: Binary Consensus Algorithm

```
Input: Set of Processes $\Pi$
Input: Initial Proposal Value $x_i$
Output: Decision Value $d_i$
$d_i \leftarrow \bot$;                                  // decision value;
$r_i \leftarrow 0$;                                     // round number;
$stop_i \leftarrow \bot$;       // round number where execution halts;
while do
    broadcast($2r + 1, \Pi, x_i$);                      // step 1;
    $V_i \leftarrow$ receive($2r + 1, \Pi$);
    if $\exists_{v \neq \bot} : \#_v(V_i) \geq 2f + 1$ then
        $x_i \leftarrow v$;
    else
        $x_i \leftarrow \bot$;
    end

    broadcast($2r + 2, \Pi, x_i$);                      // step 2;
    $V_i \leftarrow$ receive($2r + 2, \Pi$);
    if $\exists_{v \neq \bot} : \#_v(V_i) \geq 2f + 1$ then
        if $d_i = \bot$ then
            $d_i \leftarrow v$;
            $stop_i \leftarrow r_i + 1$;
        end
        $x_i \leftarrow v$;
    else if $\exists_{v \neq \bot} : \#_v(V_i) \geq f + 1$ then
        $x_i \leftarrow v$;
    else
        $x_i \leftarrow$ coin_flip();
    end
    if $r_i = stop_i$ then
        halt();                                         // stop execution;
    end
    $r_i \leftarrow r_i + 1$;
end
```

4.2.2 Description

The protocol presented in Algorithm 1 proceeds in rounds of two steps each and takes as input the proposal value x_i. Every process $p_i \in \Pi$ starts the protocol by initializing the round number r_i to 0, the decision result *d* to an undefined value $\bot$ indicating that no decision has been made yet, and the variable $stop_i$ to an undefined value $\bot$. Variable $stop_i$ will keep the round number where the algorithm should halt its execution.

Each process p_i then enters step 1 of the protocol (lines 5-11). It broadcasts x_i and saves the received messages in V_i (lines 5-6). If there are $2f+1$ messages with the same value *v* in V_i, then x_i is set to *v* to indicate the preference value of p_i. Otherwise, x_i is set to $\perp$ (lines 7-11).

Processes then proceed to step 2 (lines 12-28). Each process p_i broadcasts x_i and saves the arriving messages in V_i (lines 12-13). If there are $2f+1$ messages with the same value *v* in V_i, then p_i sets the decision value d to *v* and sets the variable $stop_i$ to indicate the algorithm should stop its execution at the next round (lines 14-18). Additionally, the proposal value x_i is updated to *v* (line 19). If there are not $2f+1$ messages with *v,* but at least $f+1$, then the only action taken is to update the proposal value x_i to *v* (lines 20-21). Otherwise, if none of the thresholds are observed, then x_i is set to a random value 1 or 0, each with probability 50% (lines 22-23). Finally, the algorithm checks if it should stop the execution by comparing the current round number r_i with the $stop_i$ variable (line 25). If the numbers match, the algorithm halts (line 26), otherwise it increments the current round number (line 28) and continues for another round.

The algorithm completes the execution exactly one round after making a decision. This means that an efficient implementation must not wait until the end of the protocol execution to output its decision value. It should output the decision value immediately when one is available and then continue the execution for an extra round.

4.2.3 Example Execution

The following table outlines a hypothetical execution of binary consensus with four processes. For every step the following information is provided (for each process): the value broadcasted, the values received, and the value computed (to be broadcast in the following step). The received values are presented in the form $<v_1,v_2,v_3,v_4>$, where v_1 is the value received from process p_1, v_2 is the value received from process p_2, and so on. An asterisk (*) indicates a faulty message transmission. For example, the representation $<v_1,*,v_3,v_4*>$ indicates the message transmission from process p_2 had an omission fault - an asterisk is present instead of a particular value - and the message from process p_4 had a corruption fault – an asterisk follows the value.

	p_1	p_2	p_3	p_4
	Round $r_i = 0$, step 1 (lines 4-11)			
Broadcast	1	1	0	1
Receive	$<1,1,0,1>$	$<1,1,0,1>$	$<1,1,0,0^*>$	$<1,1,0,0^*>$
Compute	$x_i = 1$ (line 8)	$x_i = 1$ (line 8)	$x_i = \perp$ (line 10)	$x_i = \perp$ (line 10)
	Round $r_i = 0$, step 2 (lines 12-24)			
Broadcast	1	1	$\perp$	$\perp$
Receive	$<1,1,\perp,\perp>$	$<1,1,\perp,\perp>$	$<1,*,\perp,0>$	$<1,1,\perp,\perp>$
Compute	$x_i = 1$ (line 21)	$x_i = 1$ (line 21)	$x_i = 1$ (line 23)	$x_i = 1$ (line 21)

Round $r_i = 1$, step 1 (lines 4-11)

Broadcast	1	1	1	1
Receive	$<1,1,1,0^*>$	$<1,1,1,0^*>$	$<1,1,1,0^*>$	$<1,1,1,0^*>$
Compute	$x_i = 1$ (line 8)	$x_i = 1$ (line 8)	$x_i = 1$ (line 8)	$x_i = 1$ (line 8)

Round $r_i = 1$, step 2 (lines 12-24)

Broadcast	1	1	1	1
Receive	$<1,1,1,\perp^*>$	$<1,1,1,\perp^*>$	$<1,1,1,\perp^*>$	$<1,1,1,\perp^*>$
Compute	$d_i = 1$ (line 16)	$d_i = 1$ (line 16)	$d_i = 1$ (line 16)	$d_i = 1$ (line 16)

The execution comprises two rounds until a decision is reached. The fault pattern in this example captures malicious behavior from process p_4 and an omission from process p_2. On step 1 of round $r_i = 0$, all processes except for p_3 propose value 1, however some of the transmissions from process p_4 result in corruptions. This can be seen as a deliberate attempt from p_4 to disrupt the correct operation of the protocol. As a result, only processes p_1 and p_2 propose value 1 in step 2. Processes p_3 and p_4 propose $\perp$, which indicates that they have no preference. In the transmissions carried out in this step, only process p_3 experiences a (omission) fault on the reception of a message from p_2. Since processes p_1, p_2, and p_4 receive $f+1$ messages with value 1, they compute 1 to be proposed for the next step. Process p_3, on the other hand, has to flip to coin to decide which value to broadcast the following step. This value turns out to be 1.

For the second round, every process proposes 1 as the initial value, however every transmission from p_4 results in a corruption fault, effectively changing the value from 1 to 0. Nevertheless, every process receives $2f+1$ messages with value 1, which result in every process proposing 1 to the next step. In step 2, again, every process receives $2f+1$ messages with value 1 despite every transmission from p_4 has a corruption fault. This is enough for every process to perform a decision on value 1.

4.2.4 Correctness Proof

This section shows that the above protocol ensures the three properties from Section 4.2.1. The proof is supported by Lemmas 1 and 2, which in essence demonstrate that the messages processes receive at each step can differ from one another in a limited way. Lemmas 3 and 4 are specific to prove the termination property (BC3).

Definition 1. Let Ω be the set of messages broadcast by all processes in $\prod$ at some step *t*, with $|\Omega| = n$.

Definition 2. Let V_i be the set of messages received by a process p_i at some step *t*, with $n - f \le |V_i| \le n$.

Lemma 1. At any communication step *t*, $|\Omega \cap V_i| \ge n - f$ is true.

Proof: If no transmission failures occur, then $\Omega = V_i$, i.e., the set of broadcasted messages is equal to the set of received messages. Since, according to the system model, transmission failures can originate at most from *f* processes per step, then at most *f* messages in V_i are subject to transmission failures (i.e., not received or received with different content). Therefore, the sets Ω and V_i have to contain at least $n - f$ messages in common: $|\Omega \cap V_i| \geq n - f$.

Corollary 1. Set V_i has at most *f* elements not contained in set Ω: $|V_i \setminus \Omega| \leq f$.

Lemma 2. Let V_i and V_j be the sets of messages received at any communication step by processes p_i and p_j, respectively, with $i \neq j$. Then, $|V_i \cap V_j| \geq n - f$.

Proof: According to the system model, transmission failures can originate at most from the same *f* processes per step. Therefore, any two sets V_i and V_j have to include at least the same $n - f$ elements, i.e., the messages not subject to transmission failures. It follows that $|V_i \cap V_j| \geq n - f$ must be true.

Corollary 2. Set V_j has at most *f* elements not contained in set V_i: $|V_j \setminus V_i| \leq f$.

BC1 (Validity). If all processes propose the same value *v*, then any process that decides, decides *v*.

Proof: Let V_i be the set of messages received by a process p_i in step 1. Since all processes propose the same value *v*, then, by definition, all messages in Ω have the same value *v*. According to Lemma 1, at least $n - f$ messages in Ω are also in V_i. Therefore, V_i includes at least $n - f$ messages with value *v*, and $\#_v(V_i) \geq n - f$. This means that all processes execute line 8 of the algorithm, setting $x_i = v$. A trivial inspection of the protocol shows that a similar reasoning applies to step 2. Here, if a process receives at least $n - f = 2f + 1$ messages with the same value *v*, then it decides *v*.

Lemma 3. If a process p_i decides *v* at round *r*, then no process broadcasts the value $v' \neq v \neq \perp$ at step 2 of round *r*.

Proof. For a process p_i to decide on a value *v* at round *r*, it means that it must have received at least $2f + 1$ messages with value $v \neq \perp$ at step 2 of round *r* (lines 14-19). This implies that some processes broadcasted the value $v \neq \perp$ at step 2 (line 12), and, hence, they must have set their proposal value to *v* at step 1 of round *r* (line 8). For a process p_i to set its proposal value to $v \neq \perp$ at step 1, it must receive at least $2f + 1$ messages with *v* at step 1 (lines 6-8). Consequently, no other process can receive more than $2f$ messages with value $v' \neq v$ at step 1 (see Corollary 2). It follows that no process at step 1 sets its proposal value to $v' \neq v \neq \perp$. Thus, no process broadcasts the value $v' \neq v \neq \perp$ at step 2 of round *r*.

BC2 (Agreement). No two processes decide differently.

Proof: The proof assumes that some process p_i decides on a value *v* at round *r*, and proceeds to show that no other process can decide on a value $v' \neq v$. A process p_i decides on a value $v \neq \perp$ if it receives at least $2f+1$ messages with *v* at step 2 of round *r* (lines 14-16). By Corollary 2, every other process receives at least $f+1$ messages with value *v* at step 2 of the same round. By Lemma 3, since p_i decides at round *r* (line 16), then no process broadcasts $v' \neq v \neq \perp$ at step 2 of this round (line 12). This implies, by Corollary 1, that no process receives more than *f* messages with value $v' \neq v \neq \perp$ at step 2 of round *r* (line 13). Consequently, the only value for which every process receives $f+1$ or more messages is *v*. Therefore, every process at step 2 of round *r* sets its proposal value to *v* (either at line 19 or 21). On round $r+1$, every process proposes the same value *v* at the beginning of step 1. According to BC1, any remaining process that needs to decide will also chose *v*. Thus, no two processes decide differently.

Lemma 4. At step 2 of any round *r*, if a process p_i sets its proposal value to *v* without resorting to a coin flip (line 23), then no other process sets its proposal value to $v' \neq v$ unless it resorts to a coin flip.

Proof: At step 1, a process p_i sets its proposal value to $v \neq \perp$ if it receives $2f+1$ messages with *v* (lines 7-11). Thus, according to Corollary 2, no process receives more than $2f$ messages with $v' \neq v$. Therefore, no other process sets its proposal value to $v' \neq v$. This implies that for step 2 every process broadcasts the same value *v* or $\perp$ (line 12) and, based on Corollary 1, no process receives more than *f* messages with a value $v' \neq v \neq \perp$. Thus, no process sets its proposal value at step 2 to $v' \neq v$ unless it resorts to a coin flip (line 23) since at least $f+1$ messages with a value *v'* are required to be received at step 2.

Lemma 5. If every process proposes the same value *v* at the beginning of any round *r*, then every process decides *v* by the end of round *r*.

Proof: Since all processes propose the same value *v* at the beginning of round *r*, then, by Lemma 1, all processes receive at least $n-f=2f+1$ messages with *v* at step 1. Therefore, all processes propose *v* for step 2 since they all executed line 8. This implies that, again by Lemma 1, all processes receive at least $n-f=2f+1$ messages with *v* at step 2. Consequently, all processes decide *v* at line 16.

BC3 (Termination). Every process decides within *r* rounds with a probability *P* that approaches 1 with the number of rounds: $\lim_{r\to\infty} P = 1$.

Proof: According to Lemma 5, if all processes propose the same value *v* at the beginning of a round, then all processes decide *v* at that round. The proof consists of showing that eventually all processes propose the same value *v*.

At step 2 of any round *r*, processes set their proposal values to be broadcasted at the beginning of round $r+1$. This value can be set deterministically (lines 14-19, 20-21) or randomly (lines 22-23). Let *D* be the set of processes that set their value deterministically and *R* the set of processes that set their value randomly. By Lemma

4, all processes in *D* set their proposal to the same value *v*. Therefore, with probability $2^{-|R|}$, all processes in *R* set their proposal value to *v* at the beginning of round $r+1$. In a worst-case scenario, where $|R| = n$, the probability of the processes not proposing the same value at the beginning of a round is $1-2^{-n}$. Thus, the probability *P* that all processes in *R* set their proposal to the same value *v* within *r* rounds is at worst $P=1-(1-2^{-n})^{r}$. Thus, $\lim_{r\to\infty} P = 1$, meaning that eventually all processes propose the same value *v* at the beginning of a round.

4.3 MULTI-VALUED CONSENSUS

The multi-valued consensus protocol allows processes to agree on a value *v* from an arbitrary domain V. The advantage of this protocol with respect to the binary consensus protocol is that it allows the proposed and agreed value to be anything, not just 1 or 0. Each process proposes a value $x_i \in$ V and they all decide on a value *v* proposed by some process or on a default value $\perp \notin$ V. The protocol also ensures that if all processes propose the same value *v*, then the decision value is *v*. Additionally, if a decision is made on a value $v \neq \perp$, then *v* was proposed by at least one process whose transmissions were not faulty. This is important to prevent decisions on values that are only proposed by malicious processes. The protocol resorts to binary consensus as an underlying primitive.

4.3.1 Properties

The protocol is defined formally by the following properties:

- **MVC1 (Validity 1).** If all processes propose the same value *v*, then any process that decides, decides *v*.
- **MVC2 (Validity 2).** If a process decides *v*, then *v* was proposed by at least $f+1$ processes or $v = \perp$.
- **MVC3 (Agreement).** No two processes decide differently.
- **MVC4 (Termination).** Every process eventually decides.

Multi-valued consensus provides the pivotal service of RAPTOR: agreement on arbitrary information despite transmission failures. It can be used, for example, to ensure that aircraft maneuvering is based on information that is consistent amongst all aircraft. This way, the possibility of conflicts is eliminated as long as the assumptions of the system model hold.

Algorithm 2: Multi-Valued Consensus Algorithm

```
Input: Initial proposal value $x_i$
Output: Decision value $d_i$
broadcast($1, \Pi, x_i$);                                   // step 1;
$V_i \leftarrow$ receive($1, \Pi$);
if $\exists_v : \#_v(V_i) \geq 2f + 1$ then
    $x_i \leftarrow v$;
else
    $x_i \leftarrow \perp$;
end
broadcast($2, \Pi, x_i$);                                   // step 2;
$V_i \leftarrow$ receive($2, \Pi$);
if $\exists_{v \neq \perp} : \#_v(V_i) \geq 2f + 1$ then
    $b_i \leftarrow 1$;
else
    $b_i \leftarrow 0$;
end
$c_i \leftarrow$ binary_consensus($b_i$);                   // step 3;
if $c_i = 1$ then
    $d_i \leftarrow v : \#_{v \neq \perp}(V_i) \geq f + 1$;
else
    $d_i \leftarrow \perp$;
end
```

4.3.2 Description

The protocol presented in Algorithm 2 is divided in three steps. Step 1 begins with every process p_i broadcasting the initial proposal value x_i and saving in V_i the received messages (lines 1-2). If a process p_i receives $2f+1$ or more messages with the same value v, then the proposal value x_i is set to v, otherwise it is set to a default value $\perp$ (lines 3-7). Next, at step 2, every process p_i broadcasts the updated proposal value x_i and saves the received messages in V_i (lines 8-9). If p_i receives $2f+1$ or more messages with a value $v \neq \perp$, then it sets its binary consensus proposal value b_i to 1, otherwise it sets it to 0 (lines 10-14). Finally, at step 3, a binary consensus instance is initiated using b_i as the proposal value and if its decision value is 1, then p_i decides on the value v that appears at least $f+1$ times in V_i (see Lemma 6). Otherwise, if the binary consensus decision value is 0, then p_i decides on the default value $\perp$ (lines 15-20).

4.3.3 Example Execution

The following table presents a hypothetical execution of multi-valued consensus with four processes. The representation follows the one described in section 4.2.3. Again, the fault pattern captures malicious behavior from process p_4.

	p_1	p_2	p_3	p_4
step 1 (lines 1-7)				
Broadcast	A	A	B	A
Receive	$<A,A,B,A>$	$<A,A,B,A>$	$<A,A,B,B^*>$	$<A,A,B,A>$
Compute	$x_i = A$ (line 4)	$x_i = A$ (line 4)	$x_i = \perp$ (line 6)	$x_i = A$ (line 4)
step 2 (lines 8-14)				
Broadcast	A	A	$\perp$	A
Receive	$<A,A,\perp,A>$	$<A,A,\perp,A>$	$<A,A,\perp,\perp^*>$	$<A,A,\perp,\perp^*>$
Compute	$b_i = 1$ (line 11)	$b_i = 1$ (line 11)	$b_i = 0$ (line 13)	$b_i = 0$ (line 13)
step 3 (lines 15-20)				
Propose	1	1	0	0
Decide (binary)	$c_i = 1$ (line 15)	$c_i = 1$ (line 15)	$c_i = 1$ (line 15)	$c_i = 1$ (line 15)
Decide (multi-valued)	$d_i = A$ (line 17)	$d_i = A$ (line 17)	$d_i = A$ (line 17)	$d_i = A$ (line 17)

The execution starts with processes p_1, p_2, and p_4 proposing the value A, and process p_3 proposing the value B. Since the transmission from p_4 to p_3 results in a corruption fault, only processes p_1, p_2, and p_4 propose the value A to step 2. Again, in this step, some of the transmissions from p_4 result in corruption faults. This allows only p_1 and p_2 to propose 1 to the binary consensus execution, which means that the result could be a decision of either 1 or 0. In this case, it is 1. The processes finally decide on A because it is the only value for which they have received $f+1$ messages in step 2.

4.3.4 Rigorous Proof

The proof is relatively straightforward. It only resorts to one lemma (Lemma 6). The lemma is similar, in essence, to Lemmas 1 and 2 from binary consensus. It states that processes can only receive a limited amount of contradictory messages and it serves to support the proof of the agreement property (MVC3). Additionally, the proof makes use of Lemmas 1 and 2 defined previously. These lemmas are also useful for multi-valued consensus because they are general lemmas that essentially state how much the sets of messages received by the processes overlap with each other and with the set of broadcasted messages.

MVC1 (Validity 1). If all processes propose the same value *v*, then any process that decides, decides *v*.

Proof: At step 1, if all processes propose the same initial value *v*, then by Lemma 1, every process receives at least $n-f=2f+1$ messages with *v* (lines 1-7). This implies that every process proposes *v* at step 2 (line 8). For a process to decide on a value $v' \neq v$, it has to receive at least $f+1$ messages with v' at step 2 (lines 16-17).

However, according to Lemma 1, this is impossible because at most f messages can have a value $v' \neq v$.

MVC2 (Validity 2). If a process decides v, then v was proposed by $f+1$ processes or $v = \perp$.

Proof: For a process to decide on $v \neq \perp$ (lines 16-17) some process must have seen $2f+1$ messages with v at step 2 (line 10). Otherwise, no process proposes 1 to the binary consensus protocol and the decision would have to be 0 (by BC1). By Lemma 1, for this threshold to be observable, at least $f+1$ processes have to propose v.

Lemma 6. If, at step 2, a process p_i receives at least $f+1$ messages with value v, then no process receives more than f messages with value $v' \neq v \neq \perp$.

Proof: By Lemma 1, at step 2, a process p_i receives at least $f+1$ messages with value v, if and only if some process broadcasts a message with value v (lines 8-9). If a process broadcasts a message with value v at step 2, then it is because it received at least $2f+1$ messages with value v at step 1 (lines 2-4). By Lemma 2, this implies that no process could have received more than $2f$ messages with value $v' \neq v$ at step 1 and, consequently, no process could have set its proposal value at step 1 to $v' \neq v \neq \perp$. Therefore, no process proposes $v' \neq v$ at step 2. This, according to Lemma 1, implies that no process receives more than f messages with value $v' \neq v \neq \perp$ at step 2.

MVC3 Agreement. No two processes decide differently.

Proof: Processes either decide on a value $v \neq \perp$ or decide $\perp$. These two cases are treated separately in the proof.

For the first case, a process p_i decides on a value $v \neq \perp$ if it gets 1 from the binary consensus protocol. According to the binary consensus properties (BC1, BC2, BC3), if p_i decides 1, then all processes decide 1. Hence, all processes decide on a value $v \neq \perp$ that is received at least $f+1$ times at step 2 (line 17). This value v is necessarily the same for all processes. This happens because for processes to get 1 from binary consensus, then some process had to propose 1; otherwise the decision value would had to be 0 (see BC1). A process proposes 1 to binary consensus if and only if it receives at least $2f+1$ messages with value $v \neq \perp$ at step 2 (lines 10-14). By Corollary 2, every other process has to receive at least $f+1$ messages with value $v \neq \perp$. Furthermore, according to Lemma 6, if p_i receives at least $f+1$ messages with $v \neq \perp$ at step 2, then no processes receives more than f messages with $v' \neq \perp$. Thus, if some process gets 1 from the binary consensus, every process decides on the same value $v \neq \perp$.

For the second case, a process p_i decides $\perp$ if it decides 0 on the binary consensus protocol. According to the binary consensus properties (BC1, BC2, BC3), if p_i decides 0, then all processes decide 0. Therefore, if p_i decides $\perp$, then all processes decide $\perp$.

MVC4 Termination. Every process eventually decides.

Proof: The proof is obtained from a trivial inspection of the protocol.

4.4 TERMINATING RELIABLE BROADCAST

The terminating reliable broadcast protocol is an information dissemination protocol that provides strong properties. It allows a sender process to transmit a message with value *m* that is delivered uniformly at all processes: every process is either delivered *m* or delivered a default value $\perp$. In both cases, processes are always delivered a message even if the sending process is affected by transmission failures. In this situation, processes end up getting a message $\perp$ not actually broadcast by the sending process.

4.4.1 Properties

The formal properties of the protocol are defined as follows:

- **TRB1 (Termination).** Every process is delivered exactly one message.
- **TRB2 (Validity).** If a sender process p_i broadcasts a message with value *m* and no transmissions of p_i are faulty, then p_i is delivered *m*.
- **TRB3 (Agreement).** If a process is delivered a message *m*, then all processes are delivered the same message *m*.
- **TRB4 (Integrity).** If a process is delivered a message *m*, then the message was broadcasted from the sender process p_i or $m = \perp$.

Algorithm 3: Terminating Reliable Broadcast Algorithm

```
Input: Sender Process ID $sid$
Input: Message to be Broadcast $m$          // if sender process;
Output: Delivered Message $d$
1 if $p_i = sid$ then
2     broadcast(1, $\Pi$, $m$);
3 end
4 $x_i \leftarrow$ receive(1, $\{sid\}$);
5 $d_i \leftarrow$ multivalued_consensus($x_i$);
```

4.4.2 Description

The protocol, presented in Algorithm 3 is rather simple. The sender broadcasts a message *m* and every process p_i saves the delivered message, if any, in x_i. In case no message is delivered, then $x_i = \perp$. Every process p_i then proposes x_i to a multi-valued consensus execution and the decision value *d* is the message that p_i consistently delivers.

4.4.3 Example Execution

The following table presents an execution of the terminating reliable broadcast protocol with four processes.

	p_1	p_2	p_3	p_4
Broadcast	m	-	-	-
Receive	m	m	m	m
Propose	$x_i = m$	$x_i = m$	$x_i = m$	$x_i = m$
Decide/Deliver	m	m	m	m

4.4.4 Rigorous Proof

The proof is quite straightforward because the underlying multi-valued consensus ensures that all processes deliver the same message.

TRB1 (Termination). Every process delivers exactly one message.

Proof: The proof is obtained by a trivial inspection of the protocol.

TRB2 (Validity). If a process p_i broadcasts a message with value *m* and no transmissions of p_i are faulty, then p_i delivers *m*.

Proof: If no transmissions of p_i are faulty then every process delivers the value $x_i = m$ (line 4). Consequently, every process proposes *m* to the multi-valued consensus protocol and, by MVC1 and MVC4, every process decides $d = m$ (line 4).

TRB3 (Agreement). If a process is delivered a message *m*, then all processes are delivered the same message *m*.

Proof: A process p_i delivers the message obtained from the execution of the multi-valued consensus protocol (line 5). Since this protocol ensures that every process decides on the same value, then if a process delivers $d = m$, all processes deliver *m*.

TRB4 (Integrity). If a process is delivered a message *m*, then the message was broadcasted from the sender process p_i or $m = \perp$.

Proof: If p_i does not broadcast a message *m*, then no process proposes *m* to the multi-valued consensus protocol. Only up to *f* addition faults may initiate a multi-valued consensus execution, however, by MVC2, this can only result in a decision being made on $\perp$. Thus, either the value delivered is broadcast by the sender p_i or $m = \perp$.

4.5 SERVICES FOR AIRBORNE SELF-SEPARATION

This section is concerned with the applicability of the RAPTOR protocols to airborne self-separation. To illustrate how RAPTOR can be used in practical situations, three services for airborne self-separation and SGT in particular are presented. Each service is directly based on one of the protocols and, for the most part, its implementation is relatively simple because most of the complexity is abstracted by the respective RAPTOR protocol. These services inherit the capabilities of RAPTOR, which means that the correct operation of the services is guaranteed even in the presence of arbitrary transmission faults as long as, at any particular communication step, they originate at less of one third of the processes.

The first service is an extension to RAPTOR that makes it more suitable for the dynamic environment of airborne self-separation. It deals with the fixed group assumption identified in Section 4.1. The solution resides on a group membership service, which allows an aircraft to continuously and reliably update the set of aircraft surrounding it and, consequently, the group(s) it belongs to.

The second service represents an SGT-specific service: rank consistency. It bridges that gap between the implicit SGT assumptions (i.e., timely and reliable exchange of information) and the environment (i.e., unreliable communication and potentially malicious aircraft) by providing aircraft within a group with consistent SGT rankings.

Finally, the third service is the view augmentation service. It enhances the basic SGT functionality, or any other conflict resolution algorithm, by expanding an aircraft's awareness beyond what its communication capabilities can directly provide.

4.5.1 Group Membership

The protocols of RAPTOR are group communication protocols. They operate under the assumption that aircraft have knowledge about the group, i.e., they are aware of the surrounding set of aircraft within the communication range of each other. It is only when aircraft know of the groups they belong to that the protocols can be properly executed.

Unfortunately, in an airborne self-separation environment, from the perspective of any one aircraft, the set of surrounding aircraft within communication range is constantly changing. The *a priori* group knowledge assumption is a safe one when aircraft are in vicinity of airports where the ground-based infrastructure can provide reliable group information to nearby aircraft - both grounded and airborne. Nevertheless, as aircraft takeoff and fly away, this infrastructural support eventually becomes unavailable. The aircraft require some way to update the group information in a decentralized fashion.

This section informally describes a solution to this problem. It presents an extension to RAPTOR that allows the system to handle the dynamism of the environment. Interestingly, the presented solution resorts to the RAPTOR protocols themselves and does not require any additional assumptions, except for each aircraft to be equipped with an unreliable aircraft detector. This is a module that detects the presence of nearby aircraft in a possibly incomplete manner (i.e., it may fail to detect some aircraft at some instants). It represents a type of abstraction thoroughly used in the distributed systems literature (Chandra & Toueg, 1996; Vora, Nesterenko, Tixeuil, &

Delaet, 2008). Commercial aircraft are usually able to provide this sort of support using mechanisms like ADS-B (Hicok & Lee, 1998).

The solution resides on an algorithm that is executed at pre-determined time intervals. It allows any existing group of aircraft to agree on a set of aircraft that is within a certain distance of any one of them and, based on this set, update their group information. The algorithm assumes that aircraft are bootstrapped with some initial group information. This can be easily achieved since aircraft begin their operation in an airport and, as stated before, the latter can support the former with this information. Updating the group information can mean partitioning the group, add new members to the group, dismantling the group altogether, etc. It all depends on the criteria used to form groups (any particular solution can be applied), which should usually be based on the geographical distribution of the aircraft. Thus, the protocol for group membership is as follows:

1) Each aircraft TRB-broadcasts[2] the information provided by its local aircraft detector module (i.e., the set of aircraft that were observable to it since the last time the algorithm was run) to the aircraft in its group.
2) Each aircraft receives the information that was TRB-broadcast by every other aircraft in its group.
3) This information is consolidated in a set *DA* that contains every aircraft that was observable by the aircraft of group $\prod$. This set is consistent amongst all aircraft in the group because of the properties of terminating reliable broadcast. Note that *DA* may contain aircraft inside and outside $\prod$.
4) The set *DA* is passed as an argument to an algorithm that forms one or more groups according to the geographical positioning of aircraft in *DA*. Note that RAPTOR requires that processes within a group to be within communication distance of each other defined by some distance *D*. Depending on the criteria used to form the groups, it may be desirable for any particular aircraft to be included in more than one group.
5) The information about the new groups is broadcast to the aircraft in *DA* that are not part of $\prod$.
6) Finally, all aircraft (both those in $\prod$ and those in *DA* that are not in $\prod$) update their group information to reflect the newly formed groups and their communication proceeds accordingly.

The previous algorithm is just a proof-of-concept and has room to be considerably optimized. For example, it is possible to use a regular broadcast instead of the more expensive terminating reliable broadcast followed by a vector consensus execution[3] (Correia, Neves, & Verissimo, 2006). Nevertheless, it serves to show how to use the RAPTOR protocols in a dynamic system.

[2] It uses the terminating reliable broadcast protocol.

[3] Vector consensus allows processes to agree on a vector composed by a subset of the proposed values. Although not defined as part of RAPTOR, vector consensus can be easily constructed from a multi-valued consensus primitive.

4.5.2 Rank Consistency

As described in Section 3, individual maneuvering decisions are taken based on a ranking of the aircraft performed by the SGT algorithm. The ranking is built based on the information exchanged among the aircraft in a given area. If this information is not consistent across all of them, it is possible for aircraft to build contradictory rankings leading to further conflict. This can happen when messages are lost, corrupted, or purposely contradictory.

The rank consistency service ensures that all aircraft build the same ranking despite some failures. To achieve this, it resorts to the multi-valued consensus protocol, thus ensuring rank consistency even if failures originate at up to *f* out of $n = 3f + 1$ aircraft. The service is rather simple: aircraft run a multi-valued consensus execution where the proposal value of each aircraft p_i is the ranking rkg_i obtained via the SGT message exchange. If the value returned by the multi-valued consensus is a ranking $R \neq \perp$, then every aircraft uses *R* as the ranking for SGT decisions (*R* is guaranteed to be the same for all aircraft). Otherwise if the value is a ranking $R = \perp$, then every aircraft resorts to its own individual ranking rkg_i, which may be inconsistent with some other aircraft.

4.5.3 View Augmentation

As discussed in Section 4.5.1, any particular aircraft p_i may belong to more than one group at any given moment. This depends on the geographical distribution of aircraft around it. While, by definition, every aircraft known by p_i is within communication range of p_i, not all of those aircraft are necessarily within communication range of each other. This forces these aircraft to be partitioned into more than one group. The view augmentation service has the potential to expand the awareness of a particular group to the groups adjacent to it (two groups are adjacent if they contain the same aircraft p_i as a member). From an airborne self-separation perspective, this information may be useful in case it is desirable to include a larger set of aircraft in any conflict resolution or optimization calculation such as the ones performed by SGT.

View augmentation of a particular group $\prod$ is performed by having any of its members to transmit the information about any other groups they might belong to $\prod$. This is achieved by resorting to the terminating reliable broadcast protocol. It ensures that every member of $\prod$ has the same information about the other groups, thus guaranteeing that every member of $\prod$ performs the calculations based on the same consistent information leading to compatible actions.

5 FUTURE TRENDS

Distributed systems, and the disciplines that come hand in hand with it, such as fault-tolerance and security, provide a readily and extremely valuable body of knowledge to be applied in the design of efficient and safe solutions for air transportation. It is expected that the fields of avionics and distributed systems further intersect in the near future. In particular, intrusion-tolerance seems very promising in bringing the desired qualities for such a critical industry like air transportation. Intrusion tolerance is concerned with building systems that are capable of delivering a correct service despite accidental faults or malicious attacks, of which critical environments such as air transportation are subject due to their complexity and exposure.

6 CONCLUSIONS

With the steady increase of air traffic volume, the demand for automated decision-support systems for ATM will grow. Decisions that are fundamentally based on human interaction will have to be progressively replaced by more efficient forms of control without adversely affecting safety. This chapter presents a dependable solution to ATM based on a distributed systems approach that, through proper modeling, introduces a set of agreement protocols to support a game theory algorithm for decentralized conflict-resolution and traffic optimization in an airborne self-separation environment. The services that arise from the application of the protocols to specific problems bridge a gap that exists between the assumptions of the game theory algorithm and the actual conditions provided by its operating environment. The overall robustness of the system is improved, increasing its resilience to deteriorating environmental conditions.

REFERENCES

Ben-Or, M. (1983). Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pp. 27-30.

Chandra, T. D., & Toueg, S. (1996). Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM , 43* (2), pp. 225-267.

Chockler, G., Demirbas, M., Gilbert, S., Newport, C., & Nolte, T. (2005). Consensus and Collision Detectors in Wireless Ad-hoc Networks. *Proceedings of the Twenty-fourth Annual ACM Symposium on Principles of Distributed Computing*, pp. 197-206.

Correia, M., Neves, N. F., & Verissimo, P. (2006). From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures. *The Computer Journal , 49* (1), pp. 82-96.

Correia, M., Verissimo, P., & Neves, N. F. (2006). Byzantine Consensus in Asynchronous Message-Passing Systems: a Survey. In *Resilience-building Technologies: State of Knowledge, RESIST Network of Excellence Deliverable D12.*

Cristian, F. (1991). Understanding Fault-Tolerant Distributed Systems. *Communications of the ACM , 34* (2), pp. 56-78.

Dimarogonas, D. V., & Kyriakopoulos, K. J. (2002). Inventory of Decentralized Conflict Detection and Resolution System in Air Traffic. *Technical Report HYBRIDGE, Deliverable D6.1.*

Drabkin, V., Friedman, R., & Segal, M. (2005). Efficient Byzantine Broadcast in Wireless Ad-hoc Networks. *Proceedings of the 35th IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 160-169.

Eurocontrol. (1999). *1999 Annual Report.*

FAA/Eurocontrol. (2001). Principles of Operation for the use of ASAS. *FAA/EUROCONTROL Cooperative R&D Report for Action Plan 1.*

Fischer, M. J. (1983). The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). *Proceedings of the International FCT-Conference on Fundamentals of Computation Theory*, pp. 127-140.

Fischer, M. J., Lynch, N. A., & Paterson, M. S. (1985). Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM , 32* (2), pp. 374-382.

Guerraoui, R., & Schiper, A. (1997). Consensus: The Big Misunderstanding. *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 183-188.

Hicok, D. S., & Lee, D. (1998). Application of ADS-B for airport surface surveillance. *Proceedings of the AIAA/IEEE/SAE 17th Digital Avionics Systems Conference*, *2*, pp. 34/1-34/8.

Johnson, F., Hill, J., Archibald, J., Frost, R., & Stirling, W. (2005). A Satisficing Approach to Free Flight. *Proceedings of the 2005 IEEE International Conference on Networking, Sensing and Control*, pp. 123-128.

Koo, C.-Y., Bhandari, V., Katz, J., & Vaidya, N. (2006). Reliable Broadcast in Radio Networks: the Bounded Collision Case. *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, pp. 258-264.

Kuchar, L., & Yang, L. (2000). A Review of Conflict Detection and Resolution Modeling Methods. *IEEE Transactions on Intelligent Transportation Systems , 1* (4), pp. 179-189.

Moniz, H., Neves, N. F., Casimiro, A., & Veríssimo, P. (2007). Intrusion-Tolerance in Wireless Environments: An Experimental Evaluation. *Proceedings of the 13th IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 357-364.

Moniz, H., Neves, N. F., Correia, M., & Veríssimo, P. (2006). Randomized Intrusion-Tolerant Asynchronous Services. *Proceedings of the 36th IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 568-577.

Nordwall, B. D. (1995, July 31). Free Flight: ATC Model for the next 50 years. *Aviation Week and Space Technology , 143* (5), pp. 38-39.

Perry, T. S. (1997, August). In Search of the Future of Air Traffic Control. *IEEE Spectrum , 34* (8), pp. 18-35.

Rabin, M. O. (1983). Randomized Byzantine Generals. *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pp. 403-409.

RTCA Task Force 3. (1995). *Final Report.*

Santoro, N., & Widmeyer, P. (1989). Time is not a Healer. *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, *349*, pp. 304-313.

Stirling, W. (2005). Social Utility Functions - Part I: Theory. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews , 35* (4), pp. 522-532.

Turek, J., & Shasha, D. (1992). The Many Faces of Consensus in Distributed Systems. *Computer , 25* (6), pp. 8-17.

Vora, A., Nesterenko, M., Tixeuil, S., & Delaet, S. (2008). Universe Detectors for Sybil Defense in Ad Hoc Networks. *INRIA Technical Report* (6529).

Wing, D. J. (2005). A Potentially Useful Role for Airborne Separation in 4D-Trajectory ATM Operations. *Proceedings of AIAA 5th Aviation, Technology, Integration, and Operations Conference.*