

Contract-Based Byzantine Resilience in Spacecraft Swarms

Michael Sievers¹

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California 91109, USA

Azad M. Madni²

Viterbi School of Engineering, University of Southern California, Los Angeles, California 90089, USA

Future spacecraft sent to explore distant exoplanets must survive dozens of years in unknown environments and likely suffer from unpredictable fault conditions. Having velocities approaching a few percent of the speed of light, these vehicles will have but a few hours of observation before moving on to the next stellar system. With such short observation times, the multi-year round-trip light time to Earth, and potentially large number of interesting targets, these spacecraft will need to autonomously take stock of their health and plan the best observations with the available resources. Several researchers have proposed maximizing observations by deploying spacecraft swarms. Acting as a system-of-systems (SoS), swarms can investigate multiple planets as well as accommodate disruptions from unknown sources of unclear significance. Flexible resiliency contracts have been proposed as a means for optimizing swarm recovery actions through probabilistic assessment of multiple response scenarios. A weakness in this approach though is the underlying assumption that a consistent opinion of health can be reached by the swarm. Unfortunately, this assumption is violated by so-called Byzantine faults that thwart establishing interactive consistency across the swarm. Consequently, the swarm cannot unambiguously determine whether or where to apply recovery responses. This paper discusses an extension to the probabilistic underpinnings of flexible resiliency contracts that mitigate the impact of Byzantine faults without requiring additional redundancy typically employed for Byzantine fault tolerance.

Nomenclature

α	=	finite set of actions
a	=	an action
b	=	belief state
A, G	=	contract assumption (pre-condition), guarantee (post-condition)
β	=	infinite set of belief states
C	=	contract
i	=	input
I	=	set of all possible inputs
FRC	=	Flexible Resiliency Contract
$\Lambda(b' b, a)$	=	Transition function
o	=	an observation
O	=	set of all possible observations
$POMDP$	=	Partially Observable Markov Decision Process
$\rho(b, a)$	=	Reward function
π	=	policy
s	=	state
σ	=	set of all possible states
U	=	expected utility

¹ Senior Systems Engineer, Flight Systems Engineering, 4800 Oak Grove Drive, Pasadena, CA 91109, M/S 126-260, AIAA Associate Fellow

² Professor, Astronautics, Department of Astronautical Engineering, Los Angeles, California, 90089, AIAA Fellow

I. Introduction

LAUNCHED into a heliocentric orbit, the Kepler Mission has been surveying a swath of our Milky Way galaxy since 2009 with the goal of finding Earth-sized planets in or near the habitable zone of their star system. Liquid water and possibly life might exist on these planets making them intriguing targets for detailed studies at much closer range. As of June 2016, Kepler has found 2327 exoplanets including rocky, Earth-sized, habitable zone planets.¹ Analysis of Kepler data indicates that there may be upward of 10 billion habitable zone planets within twelve light years of our solar system².

Currently planned missions³⁻⁶ can be expected to greatly expand the count of confirmed exoplanets in our galaxy, while proposed missions such as the High-Definition Space Telescope⁷ measure atmospheric water vapor, oxygen, methane, and other organic compounds that are potential evidence of life. Since these missions are designed for the spacecraft that orbit relatively close to Earth they can only hint at the presence of active biospheres. Exoplanet fly-bys or probes are needed for confirming active biospheres. However, there are enormous programmatic and technical challenges to overcome. Some of the major challenges include: which planets to visit; how to propel the vehicle there; how to navigate; how to communicate; who will be around dozens to a hundred or more years after launch to receive and process the data; and what to build that survives the trip and operates in what is almost certainly an unknown and hostile environment? This paper discusses a resiliency^{23,24} approach for managing Byzantine fault conditions (described later) that result from the latter challenge.

Traditional, monolithic spacecraft protected by an assortment of ad hoc, fault-avoidance and fault-tolerance methods have a high likelihood of failure in missions that are decades-long due, in large part, to the connectivity and inter-dependence of their subsystems. Moreover, monolithic spacecraft will significantly limit the number of star systems and planets visited. Spacecraft swarms have been suggested as a viable alternative to increase the probability of mission success while also enabling more exoplanet visits⁸⁻¹¹. Conceptually, a delivery vehicle deploys a spacecraft swarm when entering a stellar neighborhood. The swarm then marshals its resources for best achieving mission science objectives while the delivery vehicle continues to its next destination.

Reaching distant exoplanets within reasonable time periods necessitates achieving velocities approaching significant fractions of the speed of light. Consider, for example, exploring a star system such as Alpha Centauri which is 4.37 light-years from Earth. At 10% the speed of light, a swarm will arrive at Alpha Centauri in roughly 44 years and spend about two hours there before moving on. With approximately two hours for observation, disruptions affecting data collection will necessarily need rapid, autonomous corrective action. Autonomous and opportunistic planning will go hand-in-hand with autonomous corrective action because in all likelihood we won't know what to look at, or what observational resources are functional until the swarm is within close proximity of the star system.

A. Flexible Resiliency Contracts

Autonomous assessment of the state-of-health of a swarm and restoring functionality under unknown environmental conditions and unplanned usage bears similarity to a number of bio-inspired systems¹²⁻¹⁷. A recent paper^{18,25} looked at an artificial immune system construct that learns good and bad swarm behavior through observations made during operation. The construct is based on intelligent software agents that implement flexible resiliency contracts (FRCs). FRCs extend the concept of invariant contracts which are defined by a pair of assertions, $C = (A, G)$, in which A is an assumption (pre-condition) made on the environment and G is the guarantee (post-condition) a system makes if the assumption is met¹⁸. More precisely, invariant contracts describe a system that produces an output from set $o \in \{0_0, 0_1, \dots, 0_{o-1}\} \subseteq O$ when in the state $\sigma \in \{s_0, s_1, \dots, s_{s-1}\} \subseteq \Sigma$ for an input $i \in \{i_0, i_1, \dots, i_{i-1}\} \subseteq I$ where O is the set of all outputs, Σ is the set of all system states, and I is the set of all inputs.

Systems defined by invariant contracts are compatible with formal analyses methods that enable rigorous design and validation. However, inflexible constructs are not well-matched with unknown and unexpected disruptions that may result from unpredictable swarm environments, internal faults, usage, and interactions. Flexibility is introduced by invoking the resiliency construct shown in Fig. 1. The resiliency construct comprises iterations of: sensing the environment and system status (Sense \equiv assumption); planning actions that maximize the likelihood of achieving a goal (Plan); and executing those actions (Act \equiv guarantee). The environment and system health are sensed after each action and the planning function determines whether to

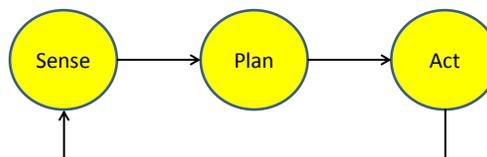


Figure 1. Resiliency implementation comprises three components: Sense, Plan, and Act.

continue with the current plan if the actions accomplish the desired outcome, or otherwise make changes.

B. Partially Observable Markov Decision Process

Flexibility is achieved through the use of Partially Observable Markov Decision Processes (POMDP) that accommodate observable, unobservable, and unknown states. A POMDP models a decision process in which system dynamics are assumed to be a belief Markovian Decision Process (MDP), a memoryless decision process that involves transition rewards. A belief MDP comprises the 4-tuple:

- β = infinite set of belief states
- α = finite set of actions
- $\rho(b, a) = \sum_{s \in S} b(s)R(s, s' | a)$ Expected reward at $b(s)$ on transition from s to s' given a
- $\Lambda(b' | b, a) = \sum_{o \in O} \Lambda(b' | b, a, o)\Lambda(o | a, b)$ Transition function

in which a *belief* represents an understanding of a system state, $s \in S$, with uncertainty. The MDP has a policy, π , which describes how to select actions for a belief state based on maximizing a goal defined by the reward function, ρ , within some time period, that is, $\pi: s \in S \rightarrow a \in \alpha$. We define the expected utility of executing π when started from s as $U^\pi(s)$. An optimal policy $\pi^* = \operatorname{argmax}_\pi U^\pi(s)$ maximizes the expected utility.

Expected utility may be computed iteratively using a number of methods including a dynamic programming approach in which a discount factor, γ , where $0 \leq \gamma < 1$, is used to penalize future rewards and is based on the “cost” for not taking immediate action. The k^{th} value of $U^\pi(s)$ is computed iteratively using Eq. (1). The optimal policy is determined by finding the policy that maximizes $U^{\pi_i}(s)$ for each policy π_i .

$$\begin{aligned}
 U_0^\pi(s) &= 0 \\
 U_1^\pi(s) &= R(s, \pi(s)) \\
 &\dots \\
 U_k^\pi(s) &= R(s, \pi(s)) + \gamma \sum_{s'} \Lambda(s' | s, \pi(s)) U_{k-1}^\pi(s')
 \end{aligned} \tag{1}$$

In the POMDP, some states are not observable (hidden) and due to imperfect information there are uncertainties regarding the current system state and the outcome of actions. The FRC approach begins with a naïve model of system behavior comprising known (designed) states and transitions as well as predicted anomalous states and transitions. The naïve model is refined over time by observing swarm behavior as actions are taken.

Post deployment, swarms perform one or more missions defined by mission scenarios. Each scenario comprises a set of mission phases further refined by a collection of detailed task behaviors allocated to the spacecraft in the swarm for accomplishing mission objectives. Mission scenarios are defined by instantiating the meta-states and transitions shown in Fig. 2. After initialization, the swarm transitions to a Planning state within mission operations. Planning takes stock of swarm health, science goals, risks, and prior knowledge of the stellar system to create an initial set of mission phases and individual spacecraft tasks. Thereafter, the swarm transitions to Cruise in which each spacecraft positions itself for making observations. Spacecraft may observe during Cruise and provide the information collected back to Planning. The swarm may also suffer from faults that require deciding how to best accomplish a mission or select a secondary mission when the primary mission is not achievable. The swarm enters Observation when it arrives at its desired locations and begins science data collection.

Fig. 2 shows that the swarm may return to Cruise between observations or may transition to degraded or unsafe operation. Fig. 2 includes restoration transitions that may return the swarm to full operational capability after a fault or may put the swarm into a degraded state by marshalling surviving swarm resources.

The swarm may return to the Planning state after concluding a mission scenario when there is sufficient time and surviving resources to carry out another mission. Additionally, Planning may be needed when restarting disrupted or aborted mission depending on when the disruption occurs during task execution and where the swarm is located relative to the observations underway. The Planning function may also modify missions as a result of “interesting” observations. For example, a spacecraft that detects water on an exoplanet may request confirmation from another spacecraft that has been tasked with observing a different exoplanet. Similarly, a spacecraft might request support from another spacecraft if a sensor needed for a particular observation has failed, or is found to be untrustworthy.

Fig. 3 shows the structure of a FRC that implements the transitions shown in Fig. 2. A belief state estimate is derived from environmental and health sensors and used for evaluating which actions to take as previously

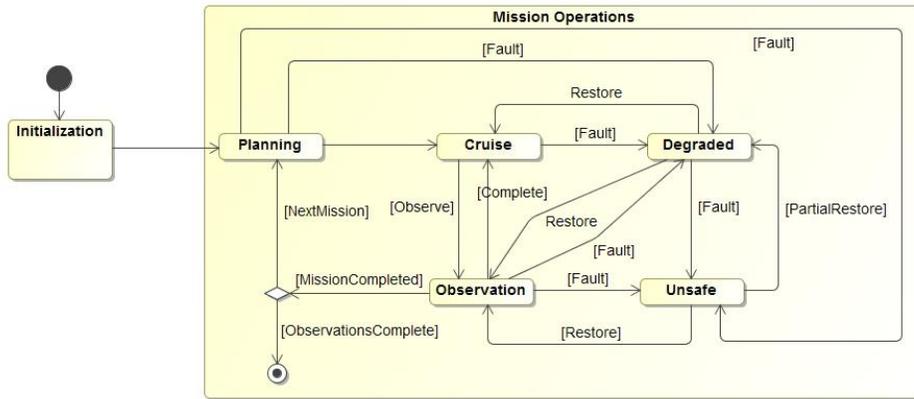


Figure 2. Mission meta-states.

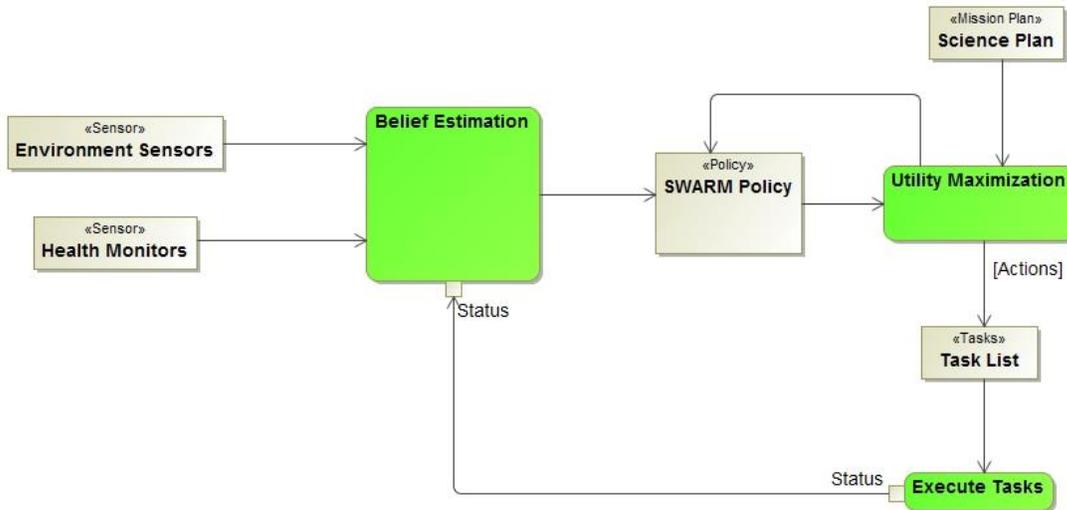


Figure 3. A belief state is estimated from input conditions and swarm execution events. The belief state is tied to the last actions taken and corresponds to the transitions in Fig. 2.

described. Actions are mapped into task lists for individual spacecraft for execution. The results of action execution are provided as feedback to the belief state estimation algorithm for evaluating the consequences of actions and determining needed new actions.

The actions taken correspond to π^* which changes depending on mission, mission phase, and tasks. For example, in Cruise the primary task for each spacecraft is safely arriving at a specified location. Small thrusters may be needed to correct trajectories but given the velocity of each vehicle, those corrections must be taken well in advance of arrival in the stellar neighborhood. If, for any reason, a spacecraft doesn't achieve the correct trajectory then policies might replan thruster firings for overcoming whatever condition caused the trajectory error, or when not possible, replan the mission to be consistent with the achievable trajectory.

Fig. 4 shows how individual FRC states are combined into an overall estimate of swarm health that triggers policies having both innate and adaptive responses. A State Estimator updates the belief state by combining observations of swarm behavior with the current belief state and the action corresponding to the maximum reward defined by the belief state and policy (Eq. 2).

$$b'(s_j) = P(s_j | o, a, b) = \frac{P(o|s_j, a) \sum_{s_i \in S} P(s_j | s_i, a) b(s_i)}{\sum_{s_j \in S} P(o|s_j, a) \sum_{s_i \in S} P(s_j | s_i, a) b(s_i)} \quad (2)$$

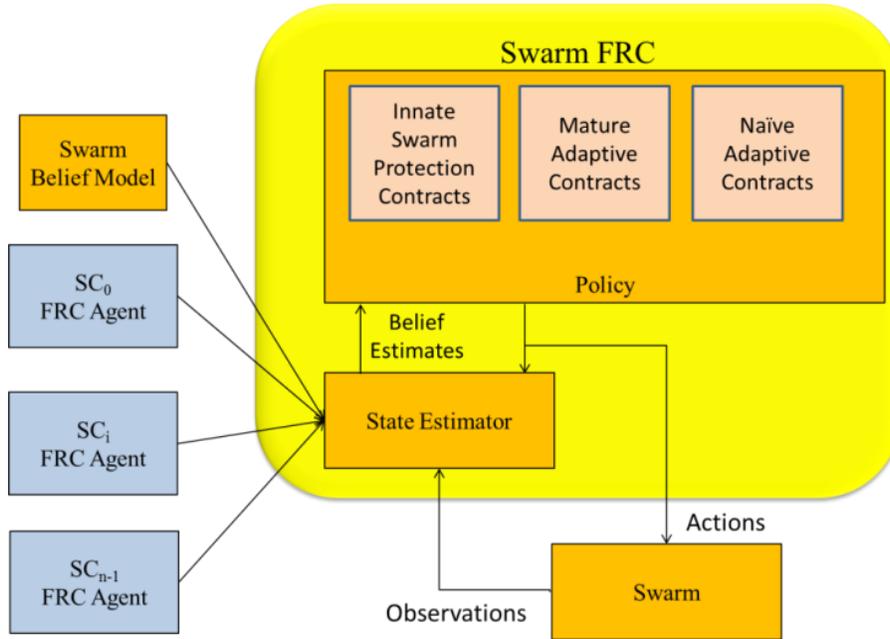


Figure 4. Swarm state blends opinions from FRC agent observations.

Innate actions are taken for fault conditions detected and verified within individual spacecraft. Innate actions correspond to invariant contracts that define local fault protection mechanisms. Adaptive responses are taken for unknown-unknowns following the pattern in Fig. 1. There are two forms of adaptive responses shown in Fig. 4 analogous to vertebrate immune responses. Naïve Adaptive Contracts (NACs) correspond to policy actions that have been randomly derived but have not yet been used. NACs persist for a finite period of time and are eventually discarded if not used. NACs become Mature Adaptive Contracts (MACs) after they have been used with positive results. Mathematically, NACs and MACs differ in the value of $R(s, \pi(s))$ and NACs are promoted to MACs when maximizing expected utility.

Figures 3 and 4 assume that establishing a consistent opinion of state is possible. But there are well-known conditions that challenge that assumption^{20, 21}.

II. Byzantine Faults

A. Interactive Consistency and Byzantine Faults

Interactive consistency occurs when all fault-free spacecraft in a swarm arrive at the same conclusion regarding overall swarm health. A Byzantine fault is one in which a faulty agent sends inconsistent information to different receivers creating the condition in which healthy spacecraft cannot agree on the state of the swarm, whether a corrective action is needed, and if needed which action to take. Lamport²⁰ defines two conditions necessary for interactive consistency: 1) all non-faulty receivers must agree on the data received from a transmitter and 2) if the transmitter is fault-free then all non-faulty receivers must also arrive at the same data value sent by the transmitter.

There are three possible fault conditions to consider: 1) a faulty spacecraft may send faulty but detectable information to fault-free receivers; 2) a faulty spacecraft may send the same faulty information to all fault-free receivers; and 3) a faulty spacecraft may send different and arbitrary information to different fault-free receivers. The first condition is easily managed simply by ignoring the faulty information. The second condition may be masked by relatively simple voting. However, the third condition is considerably more difficult to manage and is complicated by the presence of multiple faulty transmitters.

Conditions 1) and 2) result from permanent or transient faults that do not overwhelm or confuse conventional error monitors. The third condition is caused by faults that are not so severe that they cause complete spacecraft failure, but are temporally and/or spatially distributed in such a way that neither a single spacecraft nor the swarm

knows the root cause or whether and how to respond. Environmental effects are the likely culprit for condition 3) faults.

It is standard practice to provide shielding and compensation for radiation and thermal effects typically encountered by spacecraft. However, we cannot be certain that there is sufficient protection and design margin against the encountered environments or the degradations that will occur in our decades-long mission. A likely consequence will be increased rates of single-event functional interrupts (SEFI)²². SEFIs cause electronics to reset, hang, or potentially behave in unpredictable ways when affecting control functions. Resets and hangs are easily detected and managed. However, asymmetric unpredictable behaviors can potentially lead to simultaneous faulty transmitters and receivers within the swarm.

Resetting electronics affected by a SEFI generally clears the problem. However, in high radiation environments when components are potentially compromised by extended total dose damage, it is likely that most of the swarm will either be in a fault state or recovering from reset unless other measures are taken. Given that a mission might have only a few hours for making observations, this outcome is clearly not acceptable. A natural question then is whether the swarm can be designed to be sufficiently resilient so that it limps along in a high transient upset environments in which SEFIs trigger multiple, simultaneous Byzantine faults.

Nearly perfect protection against Byzantine faults is developed in Ref. 20. The solution adds sufficient system redundancy that a specified number of simultaneous Byzantine faults can be masked. The required redundancy grows with the number of faults tolerated. The swarm problem though does not necessitate instantaneous, near-perfect protection. Rather, we only need assurance that the resiliency process represented in Fig. 1 can be carried out quickly enough so that most of the mission is accomplished. That is, rather than instantaneous masking, our goal is to compute trajectories using the iteration shown in Fig. 3 that continually sample and correct swarm behavior.

B. Mitigating Byzantine Faults

There are two obvious questions resulting from the prospect of multiple Byzantine faults: 1) how can the autonomous FRC agents “know” when interactive consistency isn’t achieved and 2) what should the response be when consistency is either unknown or known to be compromised? The first question is readily answered by exchanging opinions as indicated in Fig. 3. While it may not be possible to determine which opinions to trust, it is at least possible to know that additional information is needed before taking an action. The second question is more difficult and potentially dire if the wrong decision is taken.

To begin, the following requirements are allocated to swarm communication²⁰: 1) the communication protocol assures correct delivery of all messages; messages may contain faulty information but link margins and error correcting/detecting codes guarantee that received messages are error-free; 2) receiving agents know which transmitting agent sent a message; transmitting agents will therefore need identification that cannot be forged by a faulty transmitter; 3) missing messages are detected implying some form of time-out in which the next message from a transmitting agent is expected within a timing window.

The first requirement guarantees message syntax, but not semantics. At a minimum, it assures that bit errors due to a noisy transmission channel are corrected when possible, or an uncorrectable error is detected. The former case may contribute to interactive inconsistency if the message content is faulty. The latter situation is also interesting because the faulty message is going to be ignored, effectively removing one opinion from consideration. If other messages are either semantically faulty, or contain uncorrectable errors, then there might not be enough fault-free opinions to achieve consistency.

The second requirement prevents transmitting potentially confusing messages. For example, a faulty spacecraft, *A*, sends two messages, one saying, “I’m spacecraft *A*, and I think I’m healthy and spacecraft *B* is faulty,” and the second saying, “I’m spacecraft *C*, and I think that I’m healthy and *A* is healthy.” Many options exist for implementing this requirement including simply adding unique, unforgeable identifiers to each message.

The second requirement assures timely delivery of opinions which is necessary for guaranteeing that the most current information is used in estimating state. Interestingly, simple schemes may apply that enable multiple delivery paths between spacecraft. This has the advantage of maintaining swarm connectivity when subsets of the swarm are not able to transmit to other subsets as long as all spacecraft belong to one or more subsets that can connect to the rest of the swarm.

To see how the last point might work, suppose there are three spacecraft in the swarm, *A*, *B*, and *C*. Now suppose that *A* wants to send the message, m *A*: m . Spacecraft *B* receives the message and sends it to *C* which is not in contact with *A* as *B*: A : m . *B* also sends *A* and *C* its own message, n , as *B*: n . *C* receives both messages and notices that it does not have *B*: A : m or *B*: n and adds both to its received message list. *C* then sends out its message, o , as *C*: o to both *A* and *B*, but since *A* is out-of-contact with *C*, *C*: o does not get to *A*. However, *B* receives *C*: o and sends it to *A* as

B:C:o. As *C* did before, *A* notices that it hasn't seen *B:C:o* so adds it to its receive list but knowing that it came from *C* through *B* it doesn't forward back to *B*.

Suppose there were another spacecraft, *D*, in the swarm and that *A* was in-contact with both *B* and *D*. Now suppose that *B* sends *B:n* to both *A* and *D* but the message to *A* arrives too corrupted for error correction to fix the errors. When *D* receives *B:n* though, it will send that to *A* and *C* as *D:B:n*. Since *A* does not have this message in its received list, it adds it and can use it in forming its opinion of swarm health. So the algorithm also supports multiple paths at the expense of multiple, redundant messages.

Having just defined a resilient communications method, we can now examine how to recognize and manage an inconsistency. First consider Eq. 2 that defines how the next POMDP belief state is determined after taking an action. Eq. (2) is solved for each s_j and the next action taken corresponds to the policy applied to the highest $b'(s_j)$. But suppose there isn't a clear "best" belief state or that none of the top candidates have sufficiently high probability to be trusted. In this case there is an inconsistency although it may not be obvious which FRC agent(s) are untrustworthy or what actions to take next.

Let's look at a situation in which *C*'s stellar reference unit gets confused by the star field and thinks it's in the correct location but is thousands of Km off. *B* may not know the precise location of *C* but it might be able to determine, based on its own position that *C* is in the wrong place. Suppose that *A* cannot see *C* so has no direct opinion regarding *C*'s position but based on its location it determines that *B* is likely where it should be, however, *C* thinks it is where it is supposed to be but thinks *B* is in the wrong place. Although *A* cannot see *C*, *A* knows that *C* thinks it is healthy and that *B* is faulty. *A* also knows that *B* thinks it is healthy, and that *C* is faulty. Similarly, *C* knows through *B* that *A* thinks it is healthy, and *B* agrees.

Using the delivery protocol outlined above and as represented in Fig. 4, the spacecraft exchange their belief of swarm health. Each spacecraft then creates a swarm health matrix as shown in Table 1 that shows the opinion of spacecraft in each row of the health of the spacecraft in each column. The "-" indicates "no direct opinion." From Table 1, it is evident that *A* is likely healthy but it is not clear whether *B* or *C* is faulty.

Table 1. Pairwise Health Opinions

	A	B	C
A	H	H	-
B	H	H	F
C	-	F	H

Since the pairwise opinions vindicate *A*, we use a heuristic to select which spacecraft are trustworthy and then create a reward that favors selecting opinions from *A* and penalize opinions coming from either *B* or *C*. To see how that might work, we create a MDP for our example that comprises three belief states: All Healthy (state 1), *B* Faulty (state 2), and *C* Faulty (state 3), and two actions: *take_more_measurements* (action 1) and *reset_faulty_spacecraft* (action 2) as shown in Table 2.

We compute the expected utility (Eq. 1) and the optimal policy using the *mdp_policy_iteration* function in the Matlab MDP toolbox with the results shown in Figure 5. For this example, the best course of action is to continue collecting measurements while in state 1 but reset the faulty spacecraft if either in state 2 or state 3.

Assuming each spacecraft comes to the same conclusion then *B* and *C* will reset themselves and *A* will continue mission functions. Of course it is possible that Table 1 is not identical across the swarm or that spacecraft do not agree on the action to take. For example, spacecraft *C* might decide that it doesn't need a reset. As more swarm health observations are taken then either *C* eventually forms the correct opinion of itself and resets or *A* and *B* eventually ignore *C* and replan their tasks accordingly.

Next we complicate the previous example by adding another belief state, Ambiguous (state 4) corresponding to either *B* or *C* are faulty – our Byzantine condition. We also redefine action 2 as "reset *B*," and add a new action 3, "reset *C*." Based on the pairwise comparisons shown in Table 1 we progressively assign higher rewards for the transition from state 4 to state 3 until either consistency is achieved or we discover that we are in or headed into a failed state. In the latter case, we alter our policy trajectory toward a working or safe state.

Table 2. Example MDP parameters for a 3-state, 2-action system

$\Lambda(.,.,1) =$.7 .2 .1 0 .7 .3 0 0 1.0	$R(.,1) =$	5 1 -5	$\gamma = .9$
$\Lambda(.,.,2) =$.7 .3 0 0 .8 .2 0 0 1.0	$R(.,2) =$	-1 3 5	

$U^\pi(s) =$	$\pi^*(s)$
46.5251	1
42.8571	2
50.0000	2

Figure 5. Expected utility and optimal policy for a simple 3-state, 2-action system.

The new model parameters are shown in Table 3 and the corresponding Matlab results are shown in Fig. 6 which are consistent with the observation that C should be reset in state 4. Note too that we have included transition probabilities from states 2 and 3 to state 4 that result from a reset problem causing a Byzantine fault condition. However, the primary difference between these results and the previous results are due to adjusting the reward array consistent with knowledge that A is most likely good and that A believes B is also good.

Of course it might not always be the case that there are clear comparison results that drive the reward values. When there is an equal likelihood of multiple trustworthy assets, a strategy might look at the expected utility for believing each one and then choosing the policy with the highest value of $U^\pi(s)$. If there isn't a single highest value, then the FRC should choose one and proceed until either success is achieved or another undesirable state is determined.

III. Conclusion

There are many difficult technical challenges to solve before science missions can be sent to distant exoplanets. Among these are: autonomously planning and replanning missions as a result of observations and faults; providing reliable Earth communications; autonomous navigation; propulsion; and long-term tolerance of space radiation effects. Several researchers have suggested releasing spacecraft swarms when arriving in a stellar neighborhood to maximize data collection and work-around predictable and unpredictable fault conditions that may occur. With round-trip light times on the order of years to Earth and observation times on the order of hours, spacecraft cannot simply take a safing action without risking significant loss of "once-in-a-lifetime" data.

Acting as a system-of-systems (SoS), a swarm can provide resilience to disruptions that will only be known when they happen. Resilience will comprise traditional fault-avoidance and fault-tolerance but also requires situational awareness and autonomous decision making that maximizes the chances of success. But decision making must begin with a consistent and correct understanding of the system state because errors in actions due to an incorrect assessment can lead to lost opportunities and even total mission failure.

We discussed a resiliency approach based on agents that implement Flexible Resiliency Contracts. Based on Partially Observable Markov Decision Processes (POMDP), these contracts implement "Sense-Plan-Act" by combining swarm status, state, and actions into a probabilistic belief state. The belief state is an input to a policy evaluation that maximizes expected utility for a set of actions and then chooses the action with the highest utility.

However, we noted earlier that Byzantine fault conditions could confuse a mechanistic evaluation of state and action. We have proposed a pre-processing heuristic that evaluates the trustworthiness of status information through pairwise comparisons. The heuristic increases reward values for actions related to trustworthy assets and reduces reward values for questionable assets. We have shown through an illustrative example that selecting the proper reward values lead to making correct choices. Although the heuristic has a major influence on actions taken, the iterative nature of "Sense-Plan-Act" can tolerate bad choices as long as the time to detect unsafe or tendencies toward unsafe is small compared to the point of no return. To that end, the heuristic must support minimizing the worst-case regret in taking an action. The exact nature of a suitable minimax regret algorithm is essential work-to-go and needs an analysis of decision sensitivities.

Table 3. Example MDP parameters for a 4-state, 3-action system containing a Byzantine fault.

$A(.,.,1) =$.7	.12	.12	.06	$R(.,1) =$	5	$\gamma = .9$
	0	.7	0	.3		1	
	0	0	.7	.3		1	
	0	0	0	1.0		-2	
$A(.,.,2) =$.7	.12	.12	.06		0	
	.7	.2	0	.1	$R(.,2) =$	0	
	0	0	.8	.2		1	
	0	0	0	1.0		2	
$A(.,.,3) =$.7	.12	.12	.06		0	
	0	0	.8	.2	$R(.,3) =$	0	
	.5	0	.4	.1		1	
	.5	0	.4	.1		5	

$U^\pi(s) =$	$\pi^*(s)$
39.3237	1
34.4538	2
34.6467	3
38.6467	3

Figure 6. Expected utility and optimal policy for a 4-state, 3-action system affected by a Byzantine fault.

In sum, FRCs are rigorously specified and designed resiliency agents that go hand-in-hand with autonomous mission planning and management. Our approach mimics the behavior of the invertebrate immune system by preserving safe actions and eliminating those that are either not useful or harmful. We have established the basic mathematical foundations for FRCs, and have identified future research needed to make these useful in practical applications.

References

- ¹<http://kepler.nasa.gov/>
- ²Petigura, E., Howard, A. and Marcy, G., "Prevalence of Earth-size planets orbiting Sun-like stars," *PNAS*, Vol. 110, No. 48, November 2013, pp. 19273- 19278.
- ³<http://cheops.unibe.ch/>
- ⁴http://space.mit.edu/TESS/TESS/TESS_Overview.html
- ⁵<http://www.jwst.nasa.gov/>
- ⁶<http://sci.esa.int/plato/>
- ⁷<http://www.hdstvision.org/report>
- ⁸Brown, O., and Eremenko, P., "The Value Proposition for Fractionated Spacecraft," *AIAA Space 2006*, AIAA 2006-7506, San Jose, California, September, 2006.
- ⁹Bauman, E. "Swarm-Based Concepts for Resilient Autonomous Interstellar Exploration Systems," *INCOSE INSIGHT*, Vol. 18, No. 1, pp. 34-40.
- ¹⁰Mosleh, M., Dalili, K., and Heydari, B., "Optimal Modularity for Fractionated Spacecraft: The Case of System F6," *Procedia Computer Science*, Vol 26, pp. 164-170.
- ¹¹Curtis, S., Rilee, M., Clark, P., and G. Marr, "Use of Swarm Intelligence in Spacecraft Constellations for Resource Exploration of the Asteroid Belt," Third International Workshop on Satellite Constellations and Formation Flying; Pisa, Italy, 2003:24-26
- ¹²Dorigo, M., Birattari, M., and Stützle, T., "Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique," *IEEE Computational Intelligence Magazine*, Nov. 2006, pp. 28-39.
- ¹³Montes de Oca, M.M., Stützle, T., Birattari, M., and Dorigo, M., "A Comparison of Particle Swarm Optimization Algorithms Based on Run-Length Distributions," *ANTS workshop*, 2006, pp. 1-12.
- ¹⁴Karaboga, D. and Akay, B., "A Comparative Study of Artificial Bee Colony Algorithm," *Applied Mathematics and Computation*, Vol. 214, No. 1, August 2009.
- ¹⁵Morgan, D., et. al., "Swarm-Keeping Strategies for Spacecraft under J2 and Atmospheric Drag Perturbations," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 5, Sept-Oct 2012, pp. 1492-1506.
- ¹⁶Chen, B., "Agent-Based Artificial Immune System Approach for Adaptive Damage Detection in Monitoring Networks," *Journal of Network and Computer Applications*, Vol. 33, No. 6, Nov. 2010, pp. 633-645.
- ¹⁷Chernov, A., Butakova, M., and Gorgoravo, V., "Hybrid Artificial Immune System Approach for Dynamical Agent-Based Monitoring," *Life Science Journal*; Vol. 11, 2014.
- ¹⁸Sievers, M, and Madni, A, "Agent-Based Flexible Design Contracts for Resilient Spacecraft Swarms," *AIAA SciTech*, 2016.
- ¹⁹Nuzzo, P., Sangiovanni-Vincentelli, A., Bresolin, A., Geretti, L., and Villa, T., "A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems," *Proc IEEE*, Vol. 103, No. 11, September, 2015, pp. 2104-2132.
- ²⁰Lamport, L., Shoshtak, R., and Pease, M., "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982, pp. 382-401.
- ²¹Ryan, C., Heffernan, D., and Leen, G., "Interactive Consistency on a Time-Triggered Real-Time Control Network," *IEEE Transactions on Industrial Informatics*, Vo. 2, No. 4, Nov. 2006, pp. 242-254
- ²²Mutuel, L.H., "Single Event Effects Mitigation Report," DOT/FAA/TC-15/62, February 2016.
- ²³Neches, R. and Madni, A.M. "Towards Affordably Adaptable and Effective Systems," *Systems Engineering*, Vol. 16, No. 2, pp. 224-234, Summer 2013.
- ²⁴Madni, Azad M., and Scott Jackson. "Towards a conceptual framework for resilience engineering," *IEEE Systems Journal*, 3.2, 181-191, 2009.
- ²⁵Madni, A.M. and Sievers, M. "A Flexible Contract-Based Design Framework for Evaluating System Resilience Approaches and Mechanisms," *IIE Annual Conference and Expo, ISERC 2015*, May 30- June 2, 2015.