A METHOD FOR EVALUATING AIRCRAFT ELECTRIC POWER SYSTEM SIZING AND FAILURE RESILIENCY

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Cory Kenneth Kross

January 2017

COMMITTEE MEMBERSHIP

TITLE:	A Method for Evaluating Aircraft Electric Power System Sizing and Failure Resiliency
AUTHOR:	Cory Kenneth Kross
DATE SUBMITTED:	January 2017
COMMITTEE CHAIR:	Dale Dolan, Ph.D. Associate Professor of Electrical Engineering
COMMITTEE MEMBER:	Vladimir Prodanov, Ph.D. Associate Professor of Electrical Engineering
COMMITTEE MEMBER:	Ahmad Nafisi, Ph.D. Professor of Electrical Engineering

ABSTRACT

A Method for Evaluating Aircraft Electric Power System Sizing

and Failure Resiliency

Cory Kenneth Kross

With the More Electric Aircraft paradigm, commercial commuter aircraft are increasing the size and complexity of electrical power systems by increasing the number of electrical loads. With this increase in complexity comes a need to analyze electrical power systems using new tools. The Hybrid Power System Optimizer (HyPSO) developed by Airbus SAS is a simulator designed to analyze new aircraft power systems. This thesis project will first provide a method to assess the reliability of complex aircraft electrical power systems before and after failure and reconfiguration events. Next, an add-on to HyPSO is developed to integrate the previously developed reliability calculations. Proof-of-concepts including new data visualizations are performed and provided.

Keywords: Commercial, Aircraft, Hybrid, Electrical, Power, System, Optimization, Reliability, Failure, Reconfiguration, Markov, Fault Tree Analysis, Dependency Diagram

ACKNOWLEDGMENTS

Thank you to Airbus SAS for sponsoring this project and providing financial support for my travel to France. To Dale Dolan and Vladimir Prodanov: thank you for your consistent support and advice throughout, and for your persistence and patience. A big thank-you to Cesar Antequera-Albiac for your guidance and technical support, and for hosting me during my visit to Toulouse.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Over the past century, air travel has matured into the safest mode of transportation per mile. The design of aircraft, especially those for commercial commuter purposes, are heavily scrutinized by regulatory agencies. Rightly so – a critical system failure at any point in the flight could have devastating consequences, including significant loss of life, if there are not sufficient recovery capabilities built into the aircraft. Recent technological advancements have made aircraft safer and more reliable than ever. Aircraft have become so safe that 70% of aircraft accidents leading to passenger or crew injury or death are caused by pilot error, not by system failure [1]. The odds of loss of life due to accident on any given flight, including general (non-commercial) aviation, are 1 in 4.7 million [2]. Only considering commercial aviation, that number drops to 1 in 45 million [3]. The 2013 calendar year had the fewest aviation related fatalities on record since World War II at 459 worldwide [3]. By comparison, there are forty thousand deaths per year in the United States alone related to motor vehicle accidents [4]. Figure 1 shows the downward trend in fatal accidents over time. Nevertheless, with new technologies comes increasing system complexity, and with increases in complexity come increased risk. To mitigate the risk of catastrophic failure in complex systems, the reliability of the system must be evaluated with methods that accommodate current and future technologies.

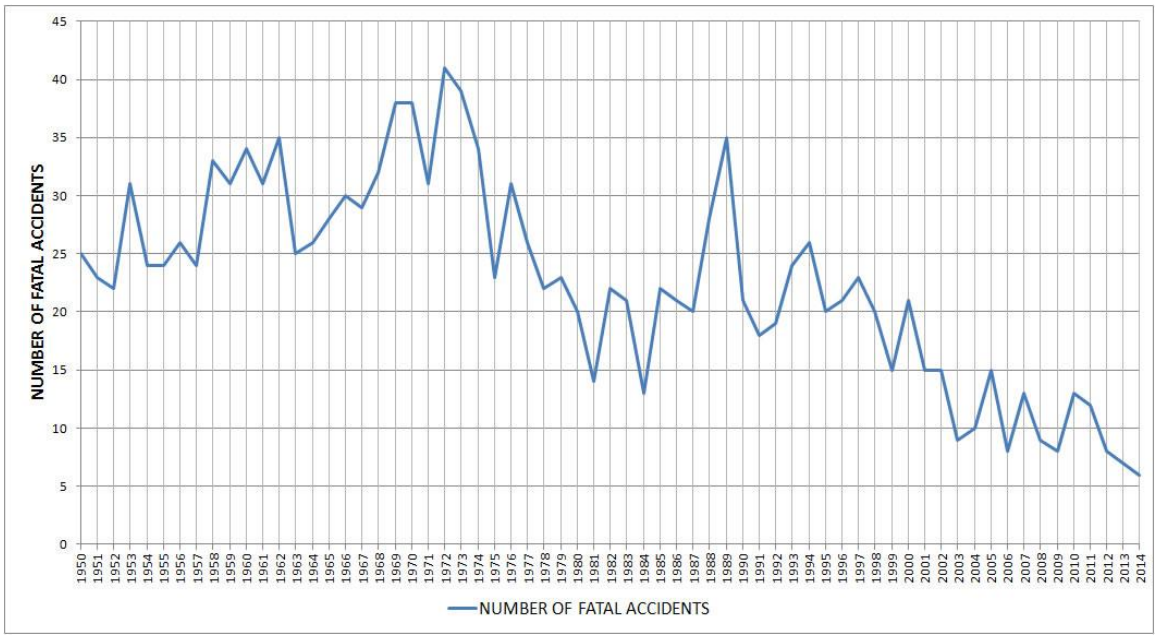


Figure 1: Fatal Accidents per Year in Civil Aircraft with 19 or More Passengers [2]

SPONSORSHIP AND SIMULATOR OVERVIEW

Airbus SAS, a leading commercial aircraft manufacturer, has sponsored this project to further research in the area of aircraft failure and reconfiguration. Airbus SAS, hereby known as the Sponsor, has provided a proprietary software simulator called Hybrid Power System Optimizer, hereafter known as HyPSO or "the Tool" for short. HyPSO performs steady-state analysis of aircraft power systems, taking into consideration mechanical power, electrical power, thermal flows and other parameters to optimize the system for minimal fuel consumption. It is a highly flexible tool, able to simulate any mission (flight plan) in varying levels of detail as required by the user. Inputs to the system include "Engine Decks" (a highly detailed, proprietary description of the fuel burn characteristics

of the aircraft engine), detailed load and machine profiles, airframe and drag characteristics, mission profiles, and atmospheric conditions. Finally, HyPSO has the capabilities to simulate failures in the aircraft, and to reconfigure the power system to recover from these failures. Simulated failures and power system reconfiguration will be the main function of the Tool exercised in this project. Most other simulation types require the Engine Deck, but this data is heavily guarded by the industry. The Sponsor is a division of Airbus Group SE, a European aerospace and defense corporation, whose primary competitors are United States aerospace corporations, including Boeing in the civil aircraft space. As the author of this project is a United States citizen, the Sponsor could only provide limited resources as per company policy. Those resources excluded Engine Decks, thereby limiting the scope of this project.

STATEMENT OF PROBLEM

The leading cost driver in commercial airliners is fuel. A medium haul aircraft (3-6 hours of flight time) will achieve a fuel efficiency anywhere from 70-100 miles per gallon per seat. A Boeing 737-400 will burn approximately 12,000 kg of fuel on a 2000 nautical mile flight – the distance from Los Angeles to New York. Fuel burn is very closely related to the mass of the aircraft. A reduction of one kilogram of mass in the aircraft will save $4,500 in a short or medium haul aircraft over twenty years of operation [5]. In a competitive commercial airliner market, there is a strong motivation to reduce operating costs, in part by reducing the weight of the aircraft. By reducing the weight of the

aircraft, the Sponsor could achieve a competitive advantage in the marketplace. In the electrical power system, weight is tied to power handling. Increasing the power consumed by the aircraft will increase the weight of the systems needed to generate and manage the power. Thus, system optimization is needed to minimize the electrical power consumption in order to reduce the weight of the power system.

Recently, the aerospace industry has been influenced by the “More Electric Aircraft” paradigm, which is driving the reduction of pneumatic, mechanical, and hydraulic power systems in favor of electrical systems. Pneumatic, mechanical, and hydraulic systems are generally mature technologies that have been in operation for many decades. Industry-wide acceptance and knowledge of these technologies contributes to widespread ability to maintain these systems. Moving to new technologies requires maintenance crews to undergo additional training, and the risk of making mistakes during maintenance increases when crews are unfamiliar with the new systems. There must be sufficient competitive advantage in new technologies for airlines to forgo mature technologies and purchase new, cutting-edge aircraft. It is difficult to say that replacing other power systems with electrical systems will reduce operating costs by increasing the overall maintainability of the aircraft; therefore, at universities and in industry, research is ongoing to determine the benefits of More Electric Aircraft with respect to fuel burn. If more research is performed on the viability and benefits of electrical systems, then the industry

may be able to develop More Electric Aircraft that better serve their customers and increase sales.

Legacy systems are mature, and there is a significant amount flight heritage and data backing up their reliability. With the move to more electrical systems, designers are relying on analytical tools such as HyPSO to show that these newer systems will be as safe and those they are replacing while improving overall system performance in terms of fuel burn. Proving that electrical systems are superior to other power types is out of the scope of this project. Instead, the objective is to add to the growing pool of research describing how a More Electric Aircraft will operate. Specifically, this project will provide a method by which the reliability of aircraft electrical power systems can be assessed using HyPSO. Using this method, the Sponsor may use the Tool to simulate the reliability of future More Electric Aircraft power systems.

OBJECTIVE

During flight, system failures may occur in the electric power system which reduces the amount of electrical power available to the system. At the system level, simulators are used to show that the electrical architecture will be able to handle all loads, even in a state of reduced power availability. Redundancy is built into electrical power distribution networks to prevent loss of power to electrical loads after system failures, thereby increasing the reliability of the electric power system. However, redundancy increases the mass of the aircraft,

thereby operating costs through increased fuel burn. There is a trade-off between increased reliability through redundancy and aircraft mass. The objective of this project is to provide a methodology and framework that aircraft electrical power systems designers may use to optimize the reliability and mass of the aircraft. The first phase of this project was the development of a MATLAB add-on to HyPSO to assist in failure simulations in conjunction with the Tool's existing capabilities. The add-on performs analysis of the large amounts of data generated by the Tool. A data structure is provided to store the simulation data, and data visualizations were built to increase the usefulness of data structure by simplifying data comprehension.

Sizing the engines and generators is a complicated problem that requires significant amounts of data such as electrical load profiles; therefore, sizing is out of the scope of the project. In the first phase of the project, a general optimization framework was built in the MATLAB add-on to assist designers in power system sizing. The second phase of the project is a proof-of-concept of the add-on, the objective of which is to increase the available power at the generators while maintaining at least the same amount of reliability in the electrical network. If the available power is increased, then either the electrical networked may be downsized to decrease weight thereby reducing operating cost, or more electrical loads may be added to improve the customer experience and allow customers to gain a competitive edge in the market.

II. BACKGROUND AND RESEARCH

AIRCRAFT POWER SYSTEMS

During normal flight operations, engines generate all power consumed throughout the aircraft. Engines burn jet fuel to generate power. Traditionally, the engine generates five types of power: thrust, electrical, pneumatic, hydraulic, and mechanical. Power plants are built into the engine to generate electrical power. A general quantitative breakdown of power generation for a conventional aircraft is shown in Figure 2. Most of the useful power extracted from the jet fuel is converted into propulsion thrust, but about 5% goes towards the other forms of power [5]. Only about 0.2% of the total engine power is consumed by electrical systems [6]. With More Electric Aircraft, the proportion of total power consumed by electrical systems is expected to increase.

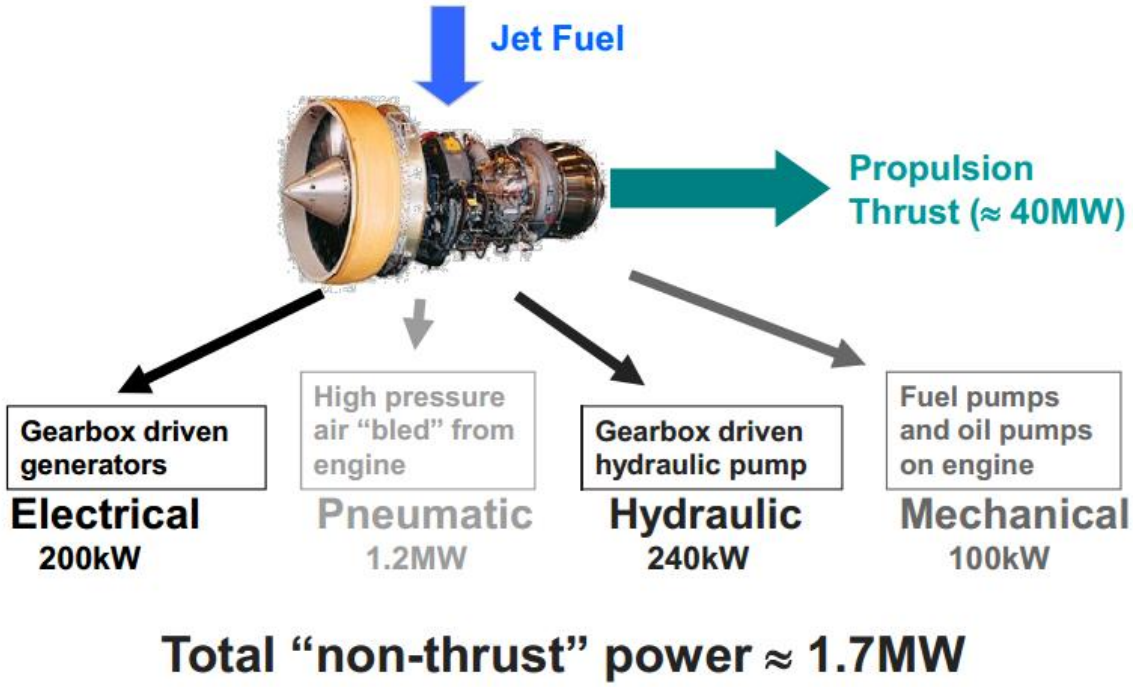


Figure 2: Breakdown of Engine Power Generation[1]

Due to the minor contribution of electrical power generation to fuel burn, electrical efficiency does not contribute significantly to fuel efficiency. However, improved electrical power efficiency does reduce waste heat generation, which is an important consideration in a closed environment such as an aircraft. Thermal management is a large topic in aircraft design, and one that HyPSO explores in detail. Reducing heat production through increased electrical efficiency will reduce the amount of weight needed for thermal management, providing another way that improved electrical systems can reduce operating costs.

Pneumatic power is high pressure air that is bled from the engine and used for wing de-icing and air conditioning, among other uses. This form of power is in the process of being phased out – the Boeing 787, Boeing's latest

commercial airliner, uses electrical heaters and compressors for de-icing and environmental control[15].

Mechanical power, used for fuel pumps and flight control surface actuation is also being phased out. Previously gearbox-driven mechanical pumps are now powered by electric motors.

Hydraulic power is generated by hydraulic pumps attached to the gearbox of the engine. Its main uses are for actuation of flight control surfaces, landing gear extension and retraction, and ground steering and braking. Engine shaft-driven hydraulic pumps are being replaced with electric pumps.

These are just a few examples of how power systems and loads are changing within the More Electric Aircraft paradigm, and the reason why power system research with respect to reliability and mass is currently in demand.

ELECTRIC POWER GENERATION

There are several common electrical power generation topologies: constant speed drive, variable speed constant frequency, and variable frequency systems, as seen in Figure 3. Conventional aircraft use one of the topologies that generate a constant 400Hz AC voltage. Due to the increase of variable frequency tolerant loads, the variable frequency technique is becoming more common. Variable frequency requires the least amount of front end mechanical structures and power electronics[8]. Two commuter aircraft that have recently reached the market, the Boeing 787 and the Airbus A380, both use this technique. The Boeing 787 has four 150kVA generators supplying four main

230V 380-800Hz distribution bus bars. Many of the architectures researched have four main electrical generators. In Appendices C and D, two Honeywell architectures are given that employ two engines with two generators per engine. The four generator topology will be the main architecture used for this project.

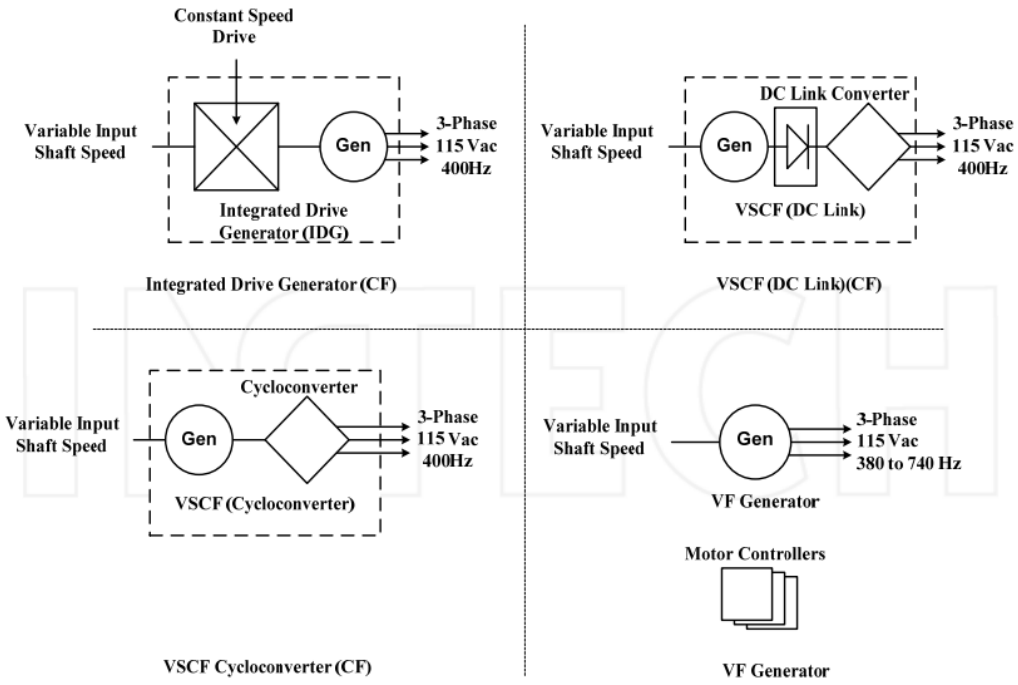


Figure 3: Electrical Power Generation Techniques [7]

When the aircraft manufacturer collaborates with an engine manufacturer, electrical power generation is of less importance than the key design point characteristics – take-off weight, cruising altitude, and thrust. Thus, the generator is bound by design decisions already made for the engine design, such as torque output and rotations per minute. Generators are designed or selected based on key parameters such as efficiency, power output during normal use, maximum power output (usually occurring in emergency situations), and time allowable at maximum power output. Data about the specific engine in use and

the load profile of the aircraft would be needed to size a generator. In particular, the engine data is highly confidential and depends greatly on the specific application. Secondly, load profiles, such as in Figure 4, require thorough testing and statistical models to describe them. Choosing a specific generator is a significant sizing and optimization problem that requires a computer simulator such as HyPSO. Generators are sized to accommodate generator failure, which is a key point that this project will explore.

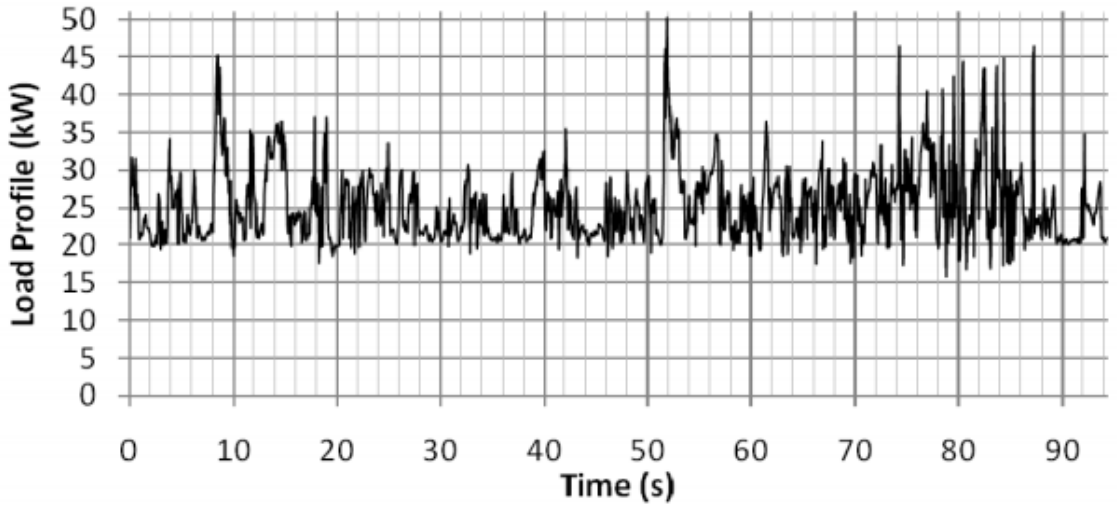


Figure 4: Sample Electrical Load Profile Over Time [3]

ELECTRIC POWER DISTRIBUTION AND MANAGEMENT

Since the electrical power generation is self-contained, the power management must also be self-contained. There are several different types of electrical loads that require different voltage types and magnitudes. For example, wing ice protection is essentially a heater, so it may utilize variable frequency AC input. Cockpit avionics are computers, so they require clean, regulated DC voltage. All of this power management occurs within electrical

panels throughout the aircraft. The Primary Electrical Power Distribution Center (PEPDC) houses the main power distribution, management, and protection devices. It is split into its redundant sections: side 1 and side 2. The PEPDC supplies the Secondary Electrical Power Distribution Centers (SEPDCs) and Secondary Power Distribution Boxes (SPDBs), which contain protective devices such as circuit breakers, contactors, and, more recently, solid state power controllers (SSPC). On the Airbus A380, there are two such SEPDCs and eight SPDBs. The Airbus A380, which first flew in 2005, has some features of the MEA trend, while retaining some conventional systems. A summary of the power distribution and management systems in the A380 is given in Table 1.

Table 1: Summary of Airbus A380 Electric Power System

4	Turbofan engines
4	150kVA, 360-800Hz AC generators
2	120kVA, 400Hz Auxiliary Power Unit (APU)
4	External Power Connections (400 Hz) for ground power
1	70kVA Ram Air Turbine
3	300A Battery charge regulator units (BCRU) (regulated Transformer Rectifier Units)
1	300A Transformer Rectifier Unit (TRU)
3	50Ah Batteries
1	Static Inverter
1	300A APU TRU (for APU starting)
1	50Ah TRU Battery (for APU starting)

The Boeing 787, first flying in 2009, is a good example of a More Electric Aircraft. The physical layout for the Boeing 787 is given in Figure 5. The PEPDC corresponds to the forward E/E bay, the SEPDC corresponds to the aft E/E bay, and the SPDBs are the remote power distribution units. It has four 250kVA

variable frequency generators. The Environmental Control System is the largest electrical load, consuming about 500kVA. Hydraulic motor pumps draw about 400kVA, and the wing ice protection system draws 100kVA[15]. A view of Boeing 787 power distribution is provided in Figure 6: Abstracted View of Boeing 787 Power Distribution [2].

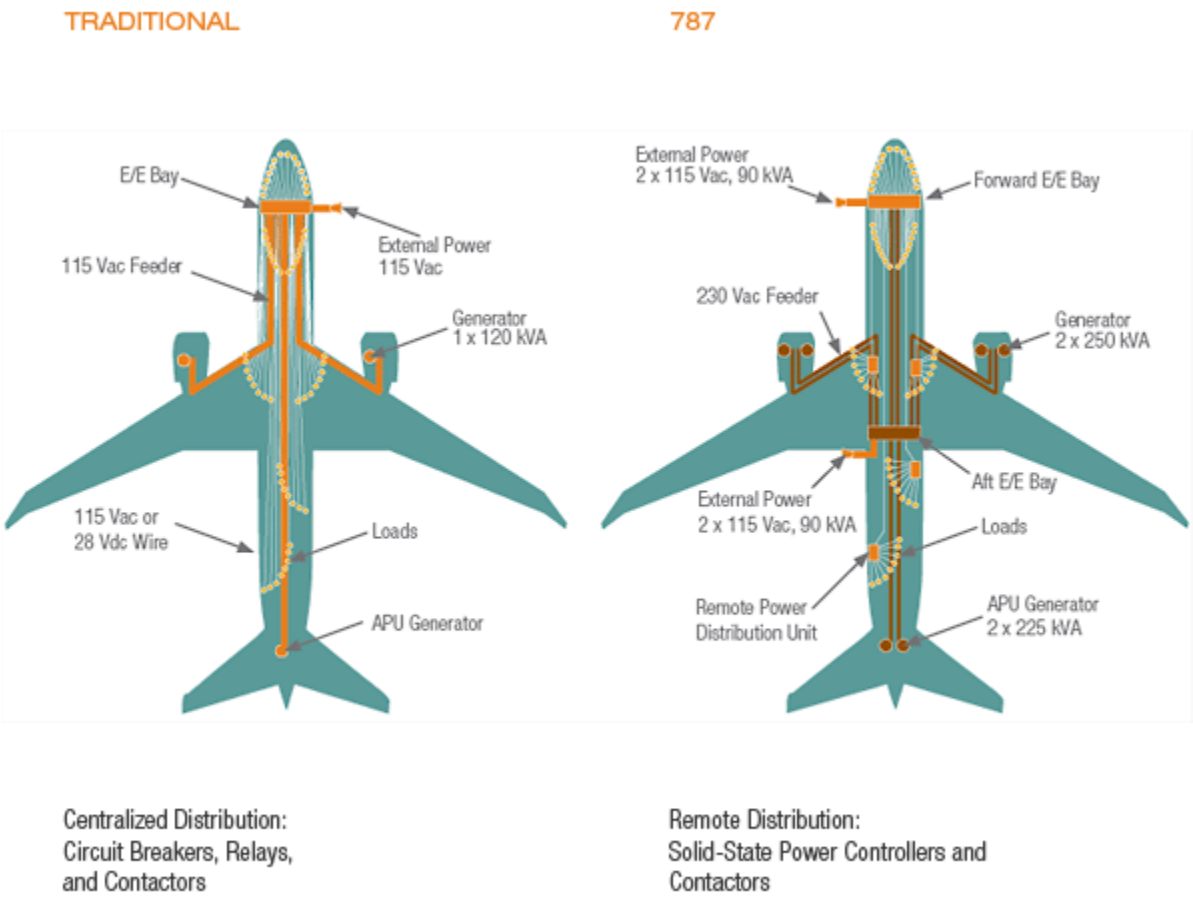


Figure 5: Physical View of Aircraft Electric Power Systems [2]

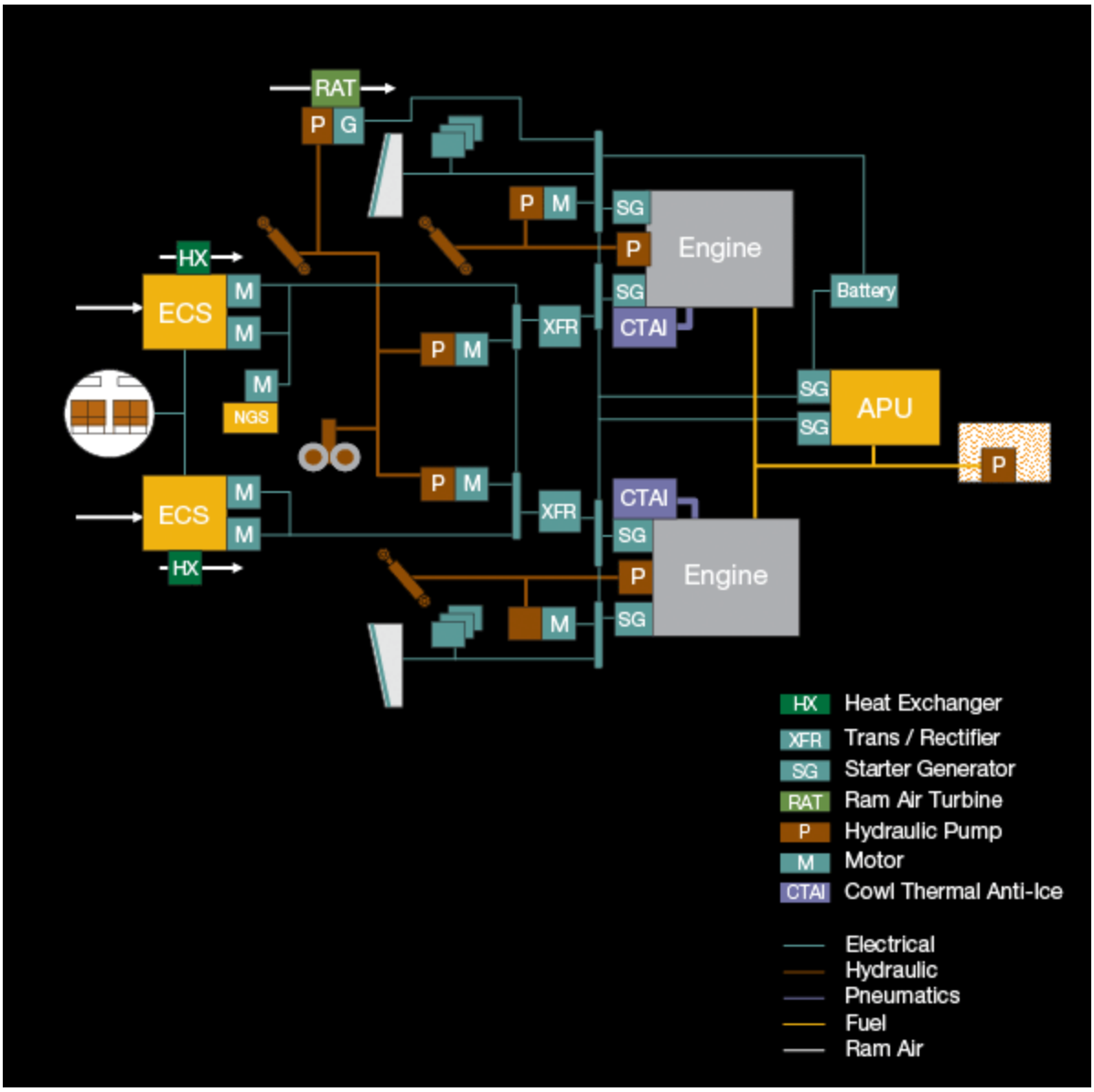


Figure 6: Abstracted View of Boeing 787 Power Distribution [2]

AIRCRAFT CERTIFICATION

The United States and the European Union have entered into the Bilateral Aviation Safety Agreement that governs civil aircraft airworthiness certification. The US is governed by the Federal Aviation Administration (FAA), and the EU by the European Aviation Safety Agency (EASA). This agreement allows simplified

certification processes in one region after the other region has already certified the aircraft. Aircraft airworthiness certification is an incredibly lengthy and rigorous process, and the aircraft design processes are developed to meet the requirements of certification. One major component to this certification is Safety Assessment [9].

Safety Assessment is performed throughout the aircraft development cycle and involves lengthy system risk and hazard assessment. Part of this assessment is the Failure Modes and Effects Analysis (FMEA), where specific failure mechanisms are identified in each system, subsystem, and component. Oftentimes FMEA involves a quantitative analysis of the failure rate of the component under examination. For the purposes of this project, it will be assumed that quantitative FMEA has been performed on each of the systems under examination. These analyses are out of the scope of this project, and the data from testing and actual system failure analysis performed in industry is unavailable due to the restrictions discussed in the Sponsorship and Simulator Overview section.

The FAA and the EASA (shown as JAA in Table 2) have defined allowable event occurrence rates for different levels of failure severities. Table 2 details the allowable failure rates per flight hours for a given failure effect. For example, a catastrophic failure, one which “prevents safe flight and landing,” must be extremely improbable or occur at a rate less than 1E-9 per flight hour, a 1 in a billion chance of occurring over one hour of flight. Failures that occur frequently, or more than once in 1000 flight hours, must have a minor severity classification,

meaning that its effect causes a “slight reduction in safety margin” or “some inconvenience to occupants.” The EASA probability classifications shall be used in this project, as summarized in Table 3.

Table 2: Failure Condition Severity as Related to Probability Objectives [9]

Probability (Quantitative)		Per flight hour 1.0	1.0E-3	1.0E-5	1.0E-7	1.0E-9
Probability (Descriptive)	FAA	Probable		Improbable		Extremely Improbable
	JAA	Frequent	Reasonably Probable	Remote	Extremely Remote	Extremely Improbable
Failure Condition Severity Classification	FAA	Minor		Major	Severe Major	Catastrophic
	JAA	Minor		Major	Hazardous	Catastrophic
Failure Condition Effect	FAA & JAA	- slight reduction in safety margins - slight increase in crew workload - some inconvenience to occupants		- significant reduction in safety margins or functional capabilities - significant increase in crew workload or in conditions impairing crew efficiency - some discomfort to occupants	- large reduction in safety margins or functional capabilities - higher workload or physical distress such that the crew could not be relied upon to perform tasks accurately or completely - adverse effects upon occupants	- all failure conditions which prevent continued safe flight and landing
Development Assurance Level	ARP 4754	Level D		Level C	Level B	Level A

Note: A "No Safety Effect" Development Assurance Level E exists which may span any probability range.

Table 3: Summary of Event Probability Classification

Probability (Qualitative)	> 1E-3	1E-3 to 1E-5	1E-5 to 1E-7	1E-7 to 1E-9	< 1E-9
Rate of Occurance	Frequent	Reasonably Probable	Remote	Extremely Remote	Extremely Improbable
Classification	Minor		Major	Hazardous	Catastrophic

III. SYSTEM RELIABILITY ANALYSIS

Reliability may be defined as the probability that no failure will occur over a given time period. Depending on the context, reliability could have different meanings for aircraft power systems. In the context of safety, a reliable aircraft is one that will not experience catastrophic system failure. Catastrophic system failures are incredibly rare, so for the purposes of this project the definition should be narrowed.

Airlines are service companies that depend on customer satisfaction to retain their business. If the customer has a bad experience on an aircraft, they will choose a different airline for their next flight, and the airline may choose a different aircraft manufacturer to improve customer experience. A bad experience could result from the failure of a non-safety related electrical load, such as the in-flight entertainment system. For a commercial aircraft, all loads are considered essential for the customer experience.

In this context, a reliable electrical power system is one that supplies all electrical loads with power during the entire flight. The reliability R of the power distribution system is calculated from P, the probability any electrical load will not be serviced with power:

$$R = 1 - P$$

Aircraft designers must maintain sufficient redundancy in the electrical network such that if any particular component fails during the flight, its redundant system may accommodate the failed component and continue to supply loads with

power. Redundancy increases the reliability of the system by decreasing the probability that a load will not be serviced.

DEPENDENCY DIAGRAMING AND FAULT TREE ANALYSIS

Two common methods of quantitative reliability analysis are dependency diagraming and fault tree analysis. These methods are used to determine the reliability of complex systems using the component failure rates determined through FMEA as previously described. The parameter of interest is the probability that any particular electrical load will be without power. The probability of failure of aircraft systems are generally given in units of flight hours; for example, the chance that a given load will be without power in any flight hour is the probability P.

A fault tree is a graphical structure that contains all of the failure modes of the system and the interrelationships of failure modes. A fault tree describes the many ways that a system can fail. This method uses traditional logic gates to structure probabilistic events. An “OR” gate is equivalent to systems in series: if one of the systems fail then the entire series chain of systems fails. For example, if an AC/DC converter fails, each load supplied by the converter also fails because it is without power. An “AND” gate is equivalent to redundant systems in parallel: all of the parallel components must fail for the system to fail. For example, if a load is supplied by two AC/DC converters, both converters must fail for the load to be without power.

A dependency diagram is a block diagram representation of a system, where the blocks are components within the system. An unbroken chain from beginning to end represents an operational system. The components have a probability of failure P, and if their failures cause the chain to be broken, then the system as a whole will fail. Figure 7 shows the correlation between a fault tree gate and the corresponding dependency diagram segment. If failure event 1 (described by P_1) occurs, then the series system represented by an OR gate will fail, but the parallel system described by the AND gate will remain functional. The parallel system described by the AND gate will fail if both failure event 1 and failure event 2 (P_2) occurs.

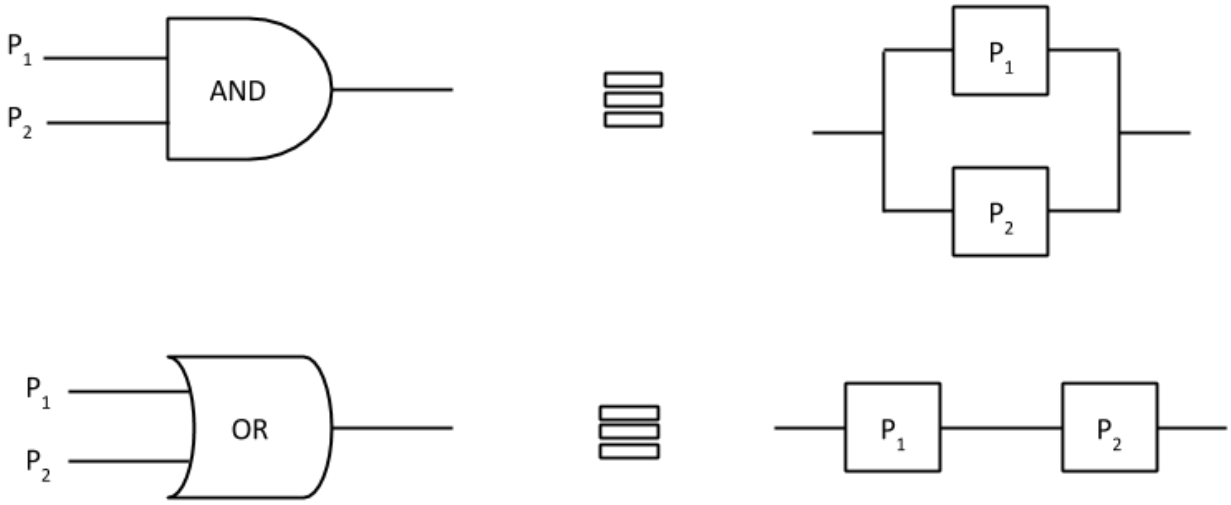


Figure 7: Fault Tree to Dependency Diagram Correspondence

This is the essence behind redundancy. By adding components in parallel, there is a backup component in case one of the components fails. By identifying and anticipating component failures, system designers can add components in parallel to increase the reliability of the system as a whole.

From the equations given next, it is apparent that adding more components in series will increase the probability that the system will fail. Adding redudant components in parallel decreases the probability that the system will fail.

$$P_{series} = 1 - \prod (1 - P_i)$$

$$P_{parallel} = \prod P_i$$

These concepts are demonstrated with a small test architecture given in Figure 8: Small Test Architecture in Nominal State with Labels. This diagram was built in yEd, a graphing tool, using a custom palette built by Airbus. Using this software, diagrams of the electrical power systems are built as inputs to HyPSO. HyPSO reads the components and interconnections in the graph and creates a description of the power system within the simulator. From there, the user inputs data describing each component. This simple test architecture will be used throughout this project as a proof of concept. There are two turbofan engines that each drive a generator. The generator supplies an AC bus, and each AC bus supplies an AC load. In reality, a bus bar supplies many loads, but for the purposes of this project the loads will be combined and represented by a single consumer of power. The AC buses also supply Transformer Rectifier Units (TRUs), which rectify the voltage to supply DC buses. Each DC bus, in turn, supplies a DC load (also a combined representation of many loads).

The small test architecture in Figure 8 is in a nominal state, meaning that it is operating normally with no failures in the system. The power paths,

represented by solid arrows, show the direction of power flow through the system. Brown arrows are mechanical power, e.g. the shaft of the engine driving an electrical generator. Green arrows are flows of electrical power. Dashed green arrows are reconfigurable power paths. In reality, a reconfigurable path is a contactor – a large switch that enables or disables a power path. In nominal states, the contactors in the reconfigurable paths are open and power is not flowing on that path. When failures occur, the contactors can close to allow power to flow along the reconfigurable path.

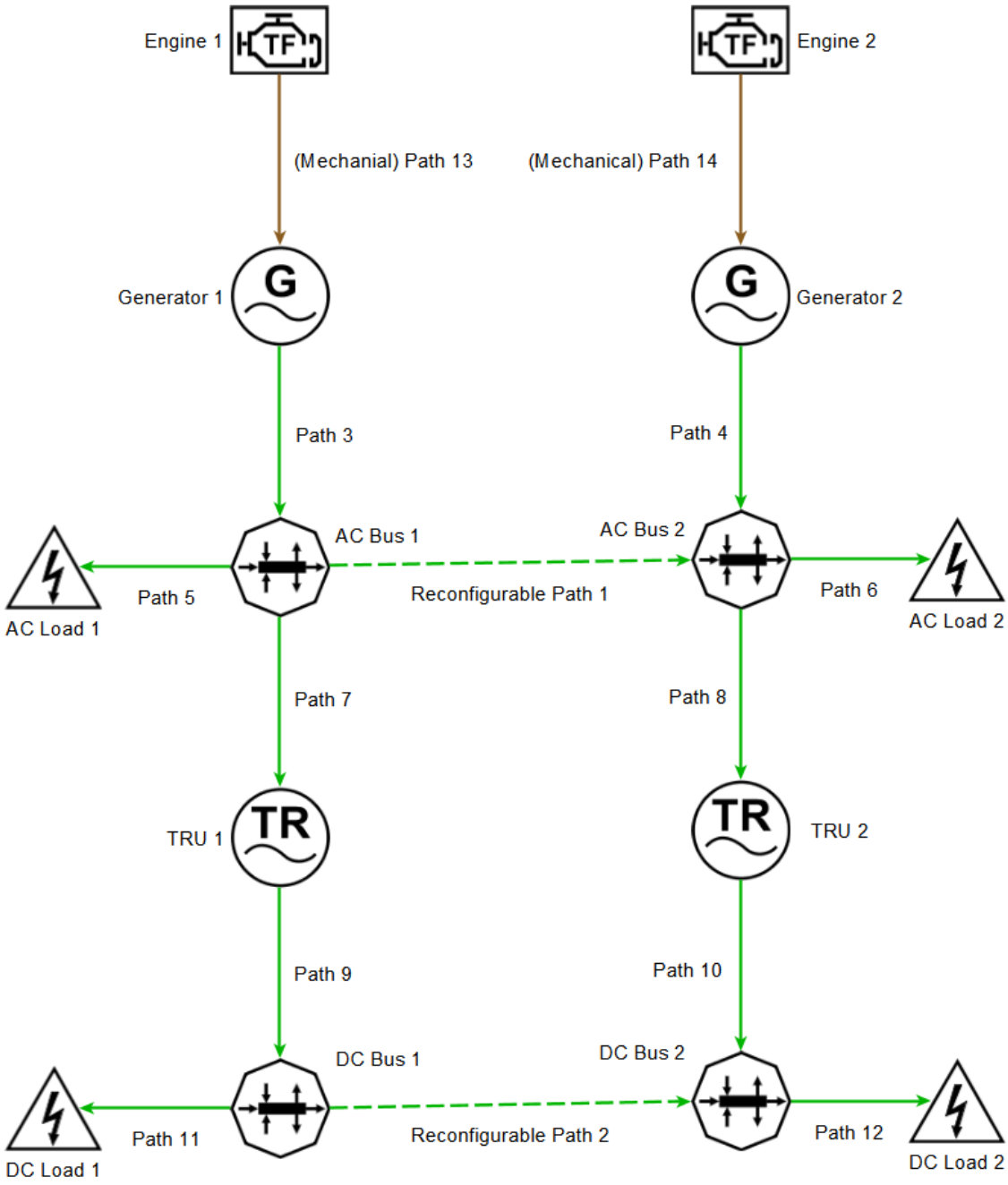


Figure 8: Small Test Architecture in Nominal State with Labels

This is a conventional, although simplified, representation of an aircraft power system. It is also within the capabilities of HyPSO to interpret the components in this architecture in various ways. The generators could be interpreted as DC generators supplying DC buses. The TRUs could be

interpreted as DC-DC converters supplying lower voltage buses, or inverters supplying AC buses. The Tool is flexible and generic in that it does not require a specification of the voltage types or magnitudes for the buses, so the user may represent them however they wish. For example, the TRU does not actually perform any voltage rectification within the Tool. A TRU simply represents the efficiency loss a TRU would have on the power system. It would also generate heat, which is not represented in Figure 8 but would be analyzed by aircraft designers using the Tool. The manufacturer of the TRU would provide the aircraft designers data for the Tool that describes the efficiency of the TRU under varying amounts of load. The output power of the TRU is less than the input power, depending on the power draw. This property applies to any voltage converter, AC-DC, DC-DC, or otherwise, thus this TRU block can represent any such converter.

Fault tree analysis and dependency diagramming were performed on this simple architecture to validate the process. For this project three modes of failure will be analyzed, but this method can and should be extended to include other failure modes. The three components set to fail are the generators, TRUs, and bus bars.

Assume the following probabilities of failure for one hour of flight (these values will be used throughout the project):

$$P_B = probability\ of\ bus\ bar\ failure = 1 * 10^{-6}$$

$$P_G = probability\ of\ generator\ failure = 7 * 10^{-4}$$

$$P_T = probability\ of\ TRU\ failure = 8 * 10^{-5}$$

In other words, the probability of a bus bar failing in a given flight hour is 1 in 1,000,000. The probability of a generator failing is 7 in 10,000 per flight hour. The probability of a TRU failing is 8 in 100,000 per flight hour.

Note for that the TRU may be treated as an electrical load of the AC bus bar. Just like the other loads attached to the bus bar, the TRU only has one input power path such that it may only be supplied by the AC bus bar. The probability that a load will fail, or not be supplied with power from the bus bar, is represented by the dependency diagram in Figure 9.

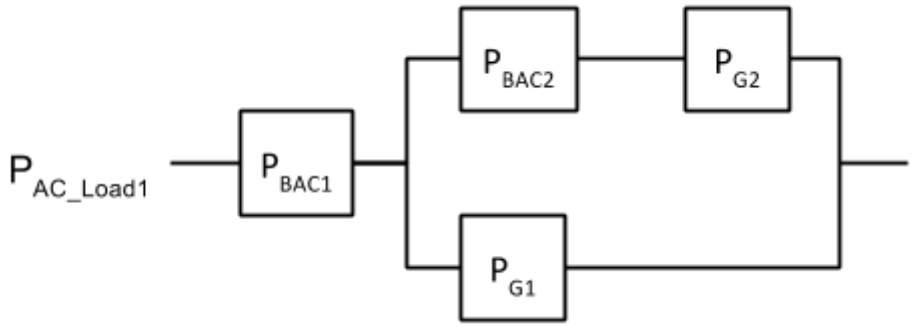


Figure 9: Dependency Diagram for AC Load 1 Failure

For AC Load 1 to fail to be supplied with power, AC Bus 1 must fail, OR Generator 1 AND Generator 2 OR AC Bus 2 must fail. If AC Bus 1 fails, there is no possible way power can reach AC Load 1, so it will fail regardless of the state of Generator 1, Generator 2, and AC Bus 2. However, if Generator 1 fails, AC Bus 1 can still be supplied with power by Generator 2. For this to occur, the contactor on Reconfigurable Path 1 will close, thereby supplying AC Bus 1 via AC Bus 2. In this failure state, if a second failure occurs in Generator 2 or Bus bar 2, no power available is from either of the two redundant paths, and AC Load 1 fails.

P_{AC_Load1} in Figure 9 is represented by the equations below. There is a 2 in a billion chance that a failure will occur in a given flight hour leading to the loss of power to AC Load 1.

$$P_{AC\_Load1} = 1 - (1 - P_{BAC1})(1 - P_{G1}[1 - (1 - P_{BAC2})(1 - P_{G2})])$$

$$P_{AC\_Load1} = 1 - (1 - 2 * 10^{-9})(1 - 1 * 10^{-9}[1 - (1 - 2 * 10^{-9})(1 - 1 * 10^{-9})])$$

$$= 2 * 10^{-9}$$

The dependency diagram in Figure 9 and the equations above also apply to AC Load 2 in a mirrored fashion by swapping first side components for second side. This scenario is shown in Figure 10. Assuming that the components in both of the redundant paths have equal probabilities of failing (e.g., P_{BAC1} = P_{BAC2}), the probability that AC Load 2 will not be powered is equal to the probability that AC Load 1 will not be powered. This parallelism between the two sides of the aircraft will be leveraged in the system reliability analysis.

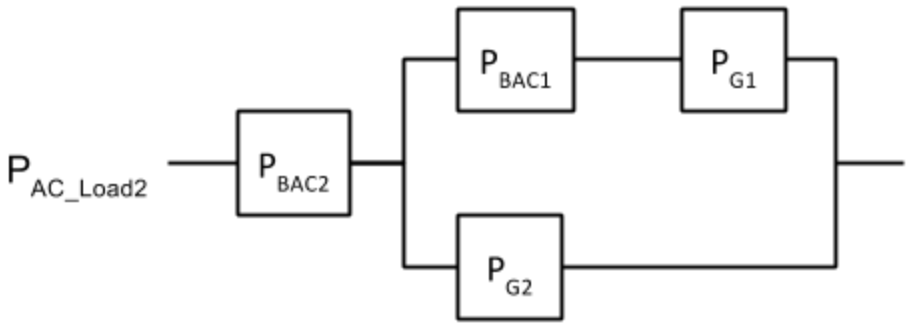


Figure 10: Dependency Diagram for AC Load 2 Failure

The probability that DC Load 1 will not be powered is described by the dependency diagram in Figure 11.

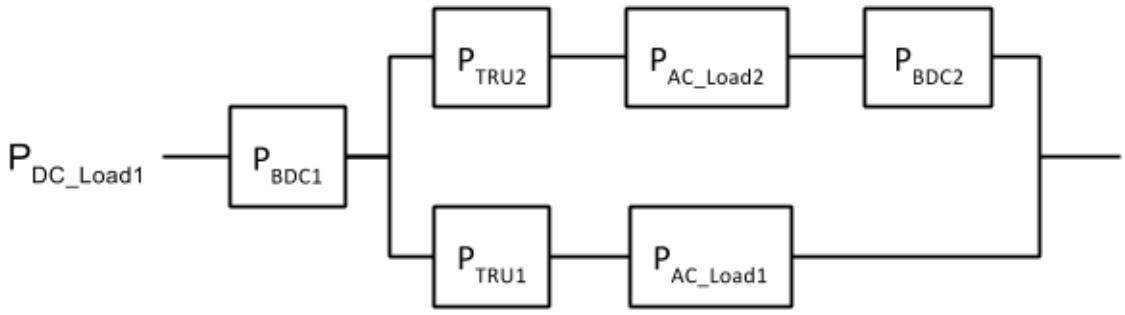


Figure 11: Dependency Diagram for DC Load Failure

This structure is slightly more complicated because there are more paths through which DC Load 1 can be supplied with power, since the load is "downstream" from both reconfigurable paths. Now both contactors may be used to route power to the load. Note that the probability of AC Load 1 and AC Load 2 being without power are included in this dependency diagram. The calculation for P_{Load_AC1} represents the probability that power cannot flow from AC Bus 1 to a load on the bus, either through normal means or through Reconfigurable Path 1. As previously mentioned, the TRU may be seen as a load of the AC bus; therefore P_{AC_Load1} also represents the probability that power is not supplied to TRU 1. The probability that DC Load 1 is without power is given below:

$$P_{DC\_Load1} = 1 - (1 - P_{BDC1})(1 - \left[1 - (1 - P_{TRU1})\left(1 - P_{AC\_Load1}\right)\right][1 - (1 - P_{BDC2})(1 - P_{TRU2})(1 - P_{AC2})]$$

$$P_{DC\_Load1} = 1 - (1 - 2 * 10^{-9})(1 - [1 - (1 - 3 * 10^{-9})(1 - 2 * 10^{-9})][1 - (1 - 2 * 10^{-9})(1 - 2 * 10^{-9})(1 - 2 * 10^{-9})])$$

$$P_{DC\_Load1} = 2 * 10^{-9}$$

The probability that any load will be without power is represented by the dependency diagram in Figure 12 and calculated below:

$$P = 1 - \left(1 - P_{AC\_Load}\right)^2 \left(1 - P_{DC\_Load}\right)^2$$

$$P = 1 - (1 - 2 * 10^{-9})^2 (1 - 2 * 10^{-9})^2$$

$$P = 8 * 10^{-9}$$

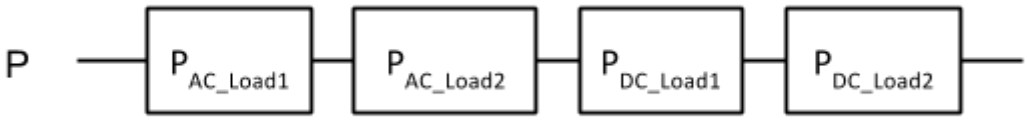


Figure 12: Dependency Diagram for Any Load Failure

Therefore, the reliability of the system is:

$$R = 1 - P = 1 - 8 * 10^{-9}$$

This result was confirmed with fault tree analysis, using the analogous representation in Figure 7. For the fault tree constructed to represent the small test architecture, refer to Appendix G. The calculation was performed in a tool developed by the Electric Power Research Institute. The tool, Computer Aided Fault Tree Analysis (CAFTA), is able to model large, complex systems with many different modes of failure. By evaluating the ANYLOAD node at the top of the fault tree in Appendix G, the tool's results match that of the calculation. The output is given in Figure 13. The probability that any load will be without power is given by the ANYLOAD output. The individual load failures are given by nodes G005 ($P_{ACLoad1}$), G011 ($P_{ACLoad2}$), G017 ($P_{DCLoad1}$), and G023 ($P_{DCLoad2}$) (see Appendix G) which each have a probability of 2*10^-9. This result matches the dependency diagrams and associated calculations.

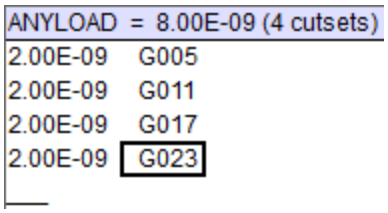


Figure 13: CAFTA Output

MARKOV ANALYSIS

The main disadvantage to using dependency diagramming or fault tree analysis to analyze a complex system such as an aircraft power system is that these methods can only evaluate a single event. For the previous example, only the probability of any load becoming unavailable was found. This method cannot describe the multitude of ways in which the distribution network can supply all loads. If an electrical generator fails, it is still possible to reconfigure the network and supply all loads with power. For highly reconfigurable aircraft power systems, a different method is needed to describe the many different configurations of the architecture, even those which have all loads supplied with power. Markov analysis is a methodology that is used to describe highly reconfigurable systems with many system states.

In Markov analysis, the system is described by “states,” which each unique state denoting a particular configuration of the system. The “state space” describes all possible states, or configurations, of a system. The state space is defined as all possible states the system can be in. Figure 14 shows an example of a Markov state space. If Figure 14 is considered a complete description of the system, then the state space S = {1 2 3 4 5}. For any given time t, the system

must be in one of the states in the state space. Thus, the probability that the system will be in one of the states in *S* at time *t* is 1, or guaranteed. The property is represented by the equation below:

$$\sum_{n=1}^{S} P_n(t) = 1$$

Transitions between states occur at a particular "rate of transition." There are five states in this state space, identified as 1 through 5, and the transition rates between states are given by λ. Note that any given state may have multiple inputs and multiple outputs.

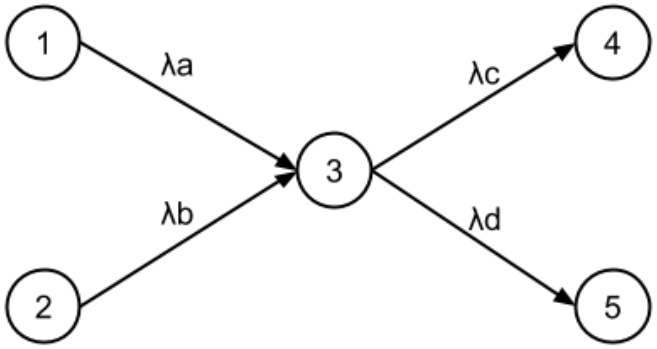


Figure 14: Example State Space with State Transition Rates

In Figure 14, the rate of transition between state 1 and state 3 is given by λ_a. Rates are given in terms of flight hours: event "a" will occur λ_a times per flight hour. A feasible value would be λ_a = 10^-4 events per flight hour. Given this rate, it is probable that event "a" will occur once every $1/\lambda_a$ flight hours, or once every 10,000 flight hours. If event "a" does occur, the system in Figure 14 will transition from state 1 to state 3. From state 3, events "c" or "d" could occur, transitioning the system to states 4 or 5, respectively.

In general, $P_n(t)$ is the probability that the system will be in state n at time t. For the system in Figure 14, the probabilities of being in each of the states are given by the differential equations below [4]. The rates of exiting a state are factored in as negatives, and the rates of entering the states are positive.

$$\frac{dP_1(t)}{dt} = -\lambda_a P_1(t)$$

$$\frac{dP_2(t)}{dt} = -\lambda_b P_2(t)$$

$$\frac{dP_3(t)}{dt} = \lambda_a P_1(t) + \lambda_b P_2(t) - (\lambda_c + \lambda_d) P_3(t)$$

$$\frac{dP_4(t)}{dt} = \lambda_c P_3(t)$$

$$\frac{dP_5(t)}{dt} = \lambda_d P_3(t)$$

By solving this system of ordinary differential equations, the probability of being in any particular state may be determined. The solution to these equations is too length to be included here. Clearly, this problem becomes highly complicated even for small state spaces. Each state has an associated ordinary differential equation, and the equations are interrelated. MATLAB is employed to solve the set of equations. The function stProb.m in Appendix E.3 was built to solve ordinary differential equations for this analysis.

In the context of aircraft electrical power systems, a state is a particular configuration of the contactors of the system. The Nominal state has the main path contactors closed, the reconfigurable path contactors open, and all

components functioning normally. The Small Test Architecture of Figure 8 is in its nominal state: all components of the system are operational, and Reconfigurable Path 1 and 2 are both open (not conducting). Markov analysis of architectures will always begin in the Nominal state. In other words, the initial condition of the system is Nominal: at time t=0 the probability of being in the nominal state is 1, or guaranteed.

$$P_{nom}(t)|_{t=0} = 1$$

From Nominal, the state space is traversed over time as failure events occur. These failure events occur at the rates given by the transition rates.

For the analysis in this project, the state space will be considered to up to two failures; therefore, two state transitions can occur. It is possible that three random failures could occur, and this analysis could easily be extended to include these cases, but the probability of three random failures occurring is so low that these events may be ignored. This brings up an important consideration regarding this analysis: a failure will be considered as independent to all other failures – meaning that a failure will not increase or decrease the chance of another failure occurring. In reality, it is feasible that a failure could propagate from one system to another, causing a second system to fail, or at least causing the rate of failure for the second system to increase dramatically.

Figure 15 shows the complete state space for up to two failures when considering the small test architecture in Figure 8. For this analysis, we will consider failures for three types of systems: generators, transformer rectifier units, and buses. In practice, this analysis should be extended to other systems

that may fail such as wiring, contactors, engines, loads, etc. Obviously, the more components that are considered, the larger the state space will be. Larger state spaces are more difficult to comprehend and manage during analysis. For a large, complex power system with many different failure modes and configurations, there could be thousands of possible states. This system of ordinary differential equations is too complicated to solve by hand. An automated tool is needed, and MATLAB is chosen to perform the calculations. In addition, this analysis will consider only the most important, top-level failure cases without regard of the actual mode of failure. A generator has many failure modes, but they will be abstracted out and one failure rate will be assigned to the generator.

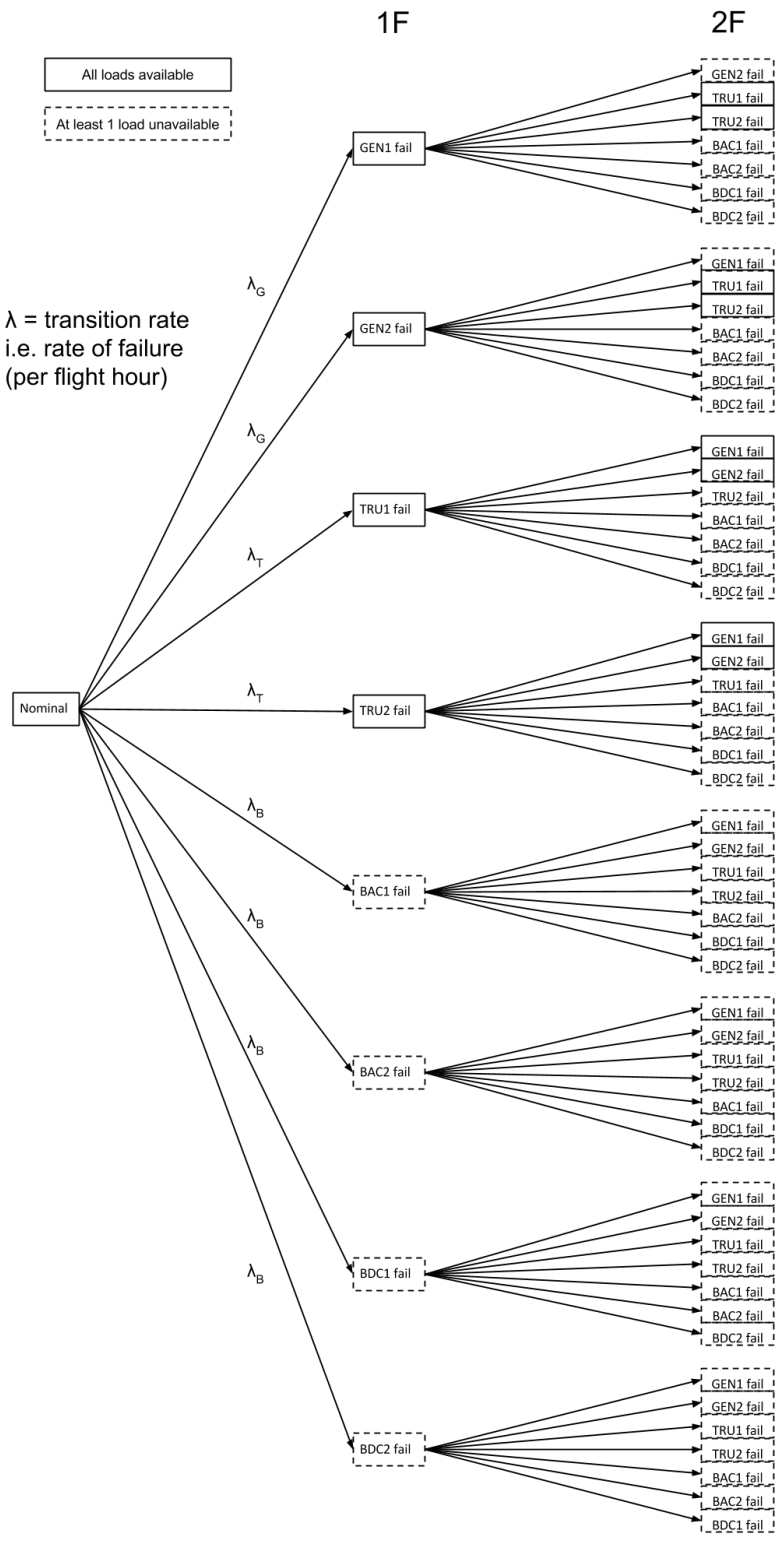


Figure 15: State Space for Small Test Architecture up to Two Failures

For the small test architecture of Figure 8, eight systems total can fail: 2 generators, 2 TRUs, and 4 buses. The state space for up to two failures is 65 states, as shown in Figure 15. Each of these states describes a configuration of the system. Each state in its particular configuration can be described in reality as having certain power flow through each component, certain power generation levels by each of the generators, and certain load availability. This is a large amount of information to store for each state. However, this information is not always unique. One state may have the same characteristics and configuration as another state. Among these 65 states, there are redundancies that allow for the state space to be simplified.

Figure 16 shows the procedure by which states may be aggregated. This example shows a case in which effectively the same state is reached by two different series of transitions. If both generators 1 and 2 fail, the order of the failures does not affect the outcome – the configuration and the power flow of the resultant state is the same whether generator 1 or 2 fails first. However, the failure rates for generator 1 and generator 2 may be different. One generator may be older than the other, or just recently undergone maintenance. The calculations below show that due to the different failure rates, P1 and P2 are different. This means that to combine identical states, each path to that state must be considered, and the probabilities must be summed together. It is not sufficient to simply multiply a state probability by the number of times an identical state occurs. In summary, the order in which systems fail DOES affect the

probability of a state occurring. However, the state configuration and parameters (available power, load availability) DO NOT depend on failure order.

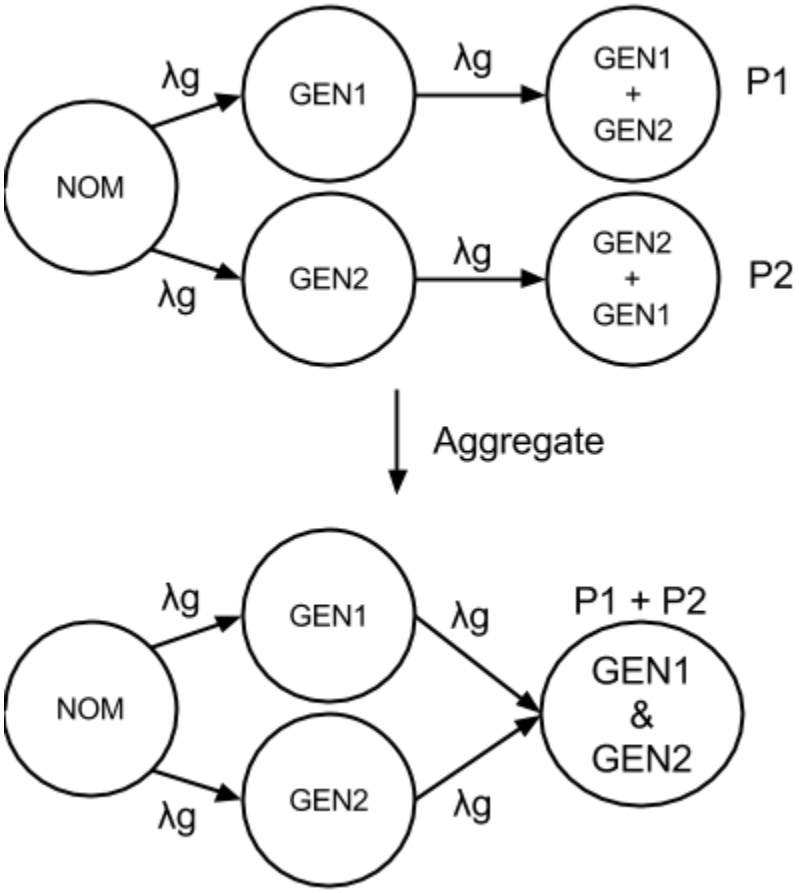


Figure 16: Example State Aggregation

Another instance in which states may be aggregated are trivial cases. Trivial cases are identified by this rule: if a bus fails, the configuration of the Reconfigurable Paths stays the same if a machine supplying, or supplied by, the bus also fails. Figure 17 illustrates this point. In this example, AC Bus 1 has failed and is removed from the architecture. There is no longer a path for Generator 1 to send power to, and no path for TRU 1 to receive power from. Therefore, if Generator 1 or TRU 1 fails, there will be no appreciable change to

the architecture configuration or parameters. In the same manner as the previous example of Figure 16, the two cases may be added to the single failure (AC Bus1) state: the case in which AC Bus1 fails then Generator 1 fails, and the case in which AC Bus 1 fails then TRU 1 fails.

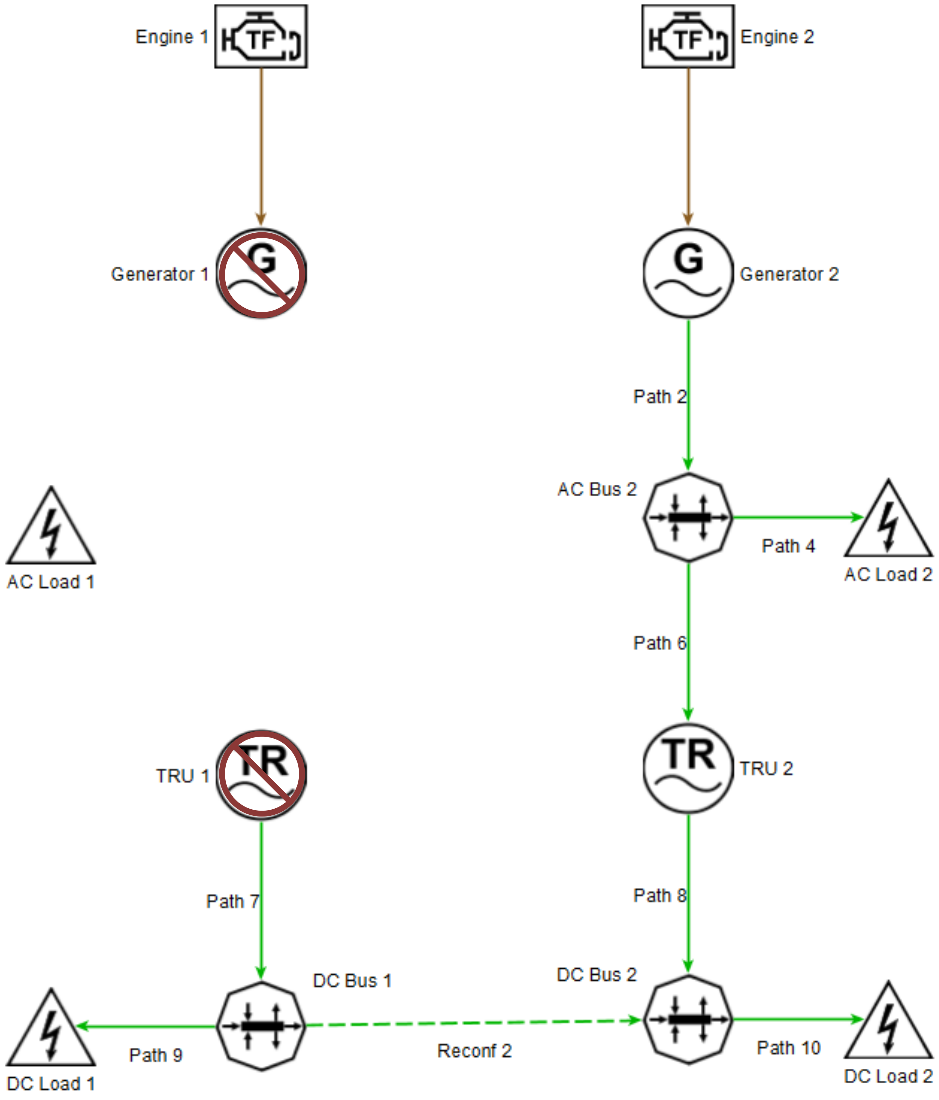


Figure 17: Example of Trivial Failure States

This is only one example of state aggregation due to trivial states. An aircraft designer performing Markov analysis could identify other rules that denote trivial states, further simplifying the state space.

IV. SIMULATIONS

The Hybrid Power System Optimizer (HyPSO) tool is a complex simulator that is capable steady-state calculations of power system parameters, including mechanical and electrical power flows, thermal generation and dissipation, and thrust. Components within the systems, such as engines, generators, and transformers, can be described to great detail using efficiency curves, thermal characteristics, minimum and maximum output power, and other parameters that are determined through testing or provided by the component manufacturers.

HyPSO SETUP

The architecture of the power system is first built in the yEd tool using a custom palette. A HyPSO setup is performed for the Full Test Architecture in Figure 18. This is a more realistic architecture than the Small Test Architecture of Figure 8. The Full Test Architecture is based on the Airbus A380. For this project, the components used will be mainly electrical power generators, distributors, converters, and consumer, but there are other types of systems available in the palette such as mechanical (e.g. gearboxes) and thermal (e.g. heatsinks). This architecture includes Essential buses and Sheddable buses, which are a common feature of aircraft electrical power systems. Essential buses must always be powered – if they are not, it could lead to a catastrophic failure of the aircraft. Sheddable buses are the opposite – in the event of an emergency, they are the first bus to be shed if there is not enough power

available to power other loads. For this analysis, we will assume that power cannot be routed through Essential and Sheddable buses to other buses.

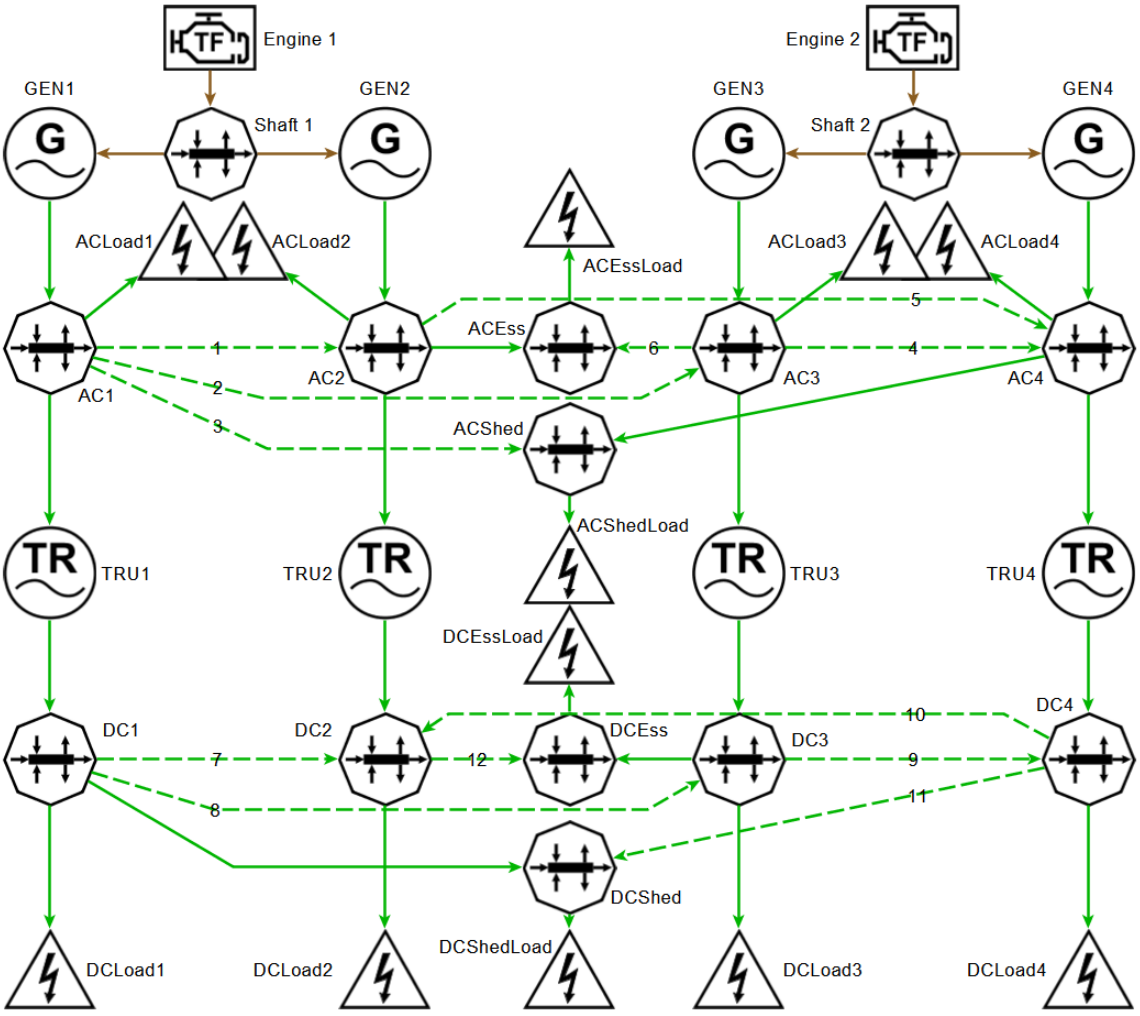


Figure 18: Full Test Architecture with Component Labels

The architecture in Figure 18 is labeled for human comprehension. The various electrical machines that will be analyzed are labeled, and electrical paths are shown in green. Reconfigurable electrical paths, or paths with a contactor that is normally open, are dashed green lines. The Tool requires that the machines, bus bars, and paths be numbered so that the Tool may interpret them

and parse them into the various tabs of the simulator. The architecture as numbered for HyPSO is provided as a reference in Appendix B.

For a very detailed system simulation, a lot of sensitive data is required. This data includes machine efficiency curves and load profiles. This data is not publicly available, so it could not be incorporated into the simulations performed for this project. Appendix F shows an example of a publicly available product sheet for a TRU – it does not include the data required to perform system simulations. Figure 19 shows the input to the Tool to represent the machines, without this sensitive data. Machines are either generators or TRU. In this example, Machines 1 through 4 are generators, and Machine 5 through 8 are TRUs. Generators are assumed to have a 100% efficiency, because the parameter of interest, available power at generators, is not affected by efficiency loss. TRUs are assumed to have a 90% efficiency. These values are represented by the Efficiency Vector. In reality, this Efficiency Vector would be variable over load. In other words, the efficiency of the machine changes with the electrical load on the machine. The other parameters, Path Out Thermal, Minimum Power, Continuous Power, Max Power, Time at Max Power, and Heat to Environment, are all parameters involved in sizing of the electrical network. As previously discussed, sizing is out of the scope of this project. In addition, these parameters were unavailable for this project.

Number of Machines: 8

	Path In	Path Out	Path Out Thermal	Min. Power [kW]	Continuous Power [kW]	Max. Power [kW]	Time at Max. Power [s]	Heat to Environment 0.00-1.00	Machine Type	Edit Vectors	Power Vector [kW]	Efficiency Vector	Interpolation Method
Machine 1	43	13	0	0	100	100	60	0	Generator	Click to Edit	[1 1000]	[1 1]	Linear
Machine 2	44	14	0	0	100	100	60	0	Generator	Click to Edit	[1 1000]	[1 1]	Linear
Machine 3	45	15	0	0	100	100	60	0	Generator	Click to Edit	[1 1000]	[1 1]	Linear
Machine 4	46	16	0	0	100	100	60	0	Generator	Click to Edit	[1 1000]	[1 1]	Linear
Machine 5	25	17	0	0	100	100	60	0	Transforme	Click to Edit	[1 1000]	[0.9 0.9]	Linear
Machine 6	26	18	0	0	100	100	60	0	Transforme	Click to Edit	[1 1000]	[0.9 0.9]	Linear
Machine 7	27	19	0	0	100	100	60	0	Transforme	Click to Edit	[1 1000]	[0.9 0.9]	Linear
Machine 8	28	20	0	0	100	100	60	0	Transforme	Click to Edit	[1 1000]	[0.9 0.9]	Linear

Figure 19: Test Architecture Machines Tab

Since engine failures are not being considered for this method, the efficiency curve of the generator is of no concern. In reality, the generator will achieve peak efficiency at a particular output, but this data is not readily available to the public. For this project, it is assumed that the engine is able to supply to the generator as much power as it needs, so the engine may be abstracted out of the problem. For all architectures under test, the generator will be able to supply a maximum of 100kW of power, with an efficiency of 100% over its entire output range. Hypothetically, if the generator was considered to have 80% efficiency while maintaining 100kW output that would simply mean that the engine would have to supply 125kW to the generator, which is possible in this scenario. The main parameter of interest, the available power at the output of the generator, is not affected by the efficiency of the generator.

When considering how to reconfigure the network after a failure, the efficiency of the TRU comes into consideration. It becomes a load balancing problem, where the efficiency could be maximized after a reconfiguration event by choosing to supply more power through one TRU than through another. This optimization problem is out of the scope of this project. It would be necessary to

have data on the efficiency curves of TRUs used in aircraft. Instead, TRUs will be considered to have a 90% efficiency over the entire output range.

The main parameters of interest in the HyPSO simulation are the available generator power and load availability. Available generator power is determined at the output routing node (bus bar) of the generator and is defined by the total power able to be supplied to the bus, minus the power consumed by loads attached to the bus, or downstream of the bus:

$$P_{available} = P_{max\,suppliable} - P_{consumed}$$

For example, in the small test architecture of Figure 8, if Generator 1 is capable of producing a maximum of 100kW of power, AC Load 1 and DC Load 1 are each consuming 10kW of power, and the efficiency of TRU 1 is 90%, then the available power at AC Bus 1 is calculated as follows:

$$P_{available(AC\ BUS\ 1)} = 100kW - \left(10kW + \frac{10kW}{90\%}\right) = 78.9kW$$

This means that at AC Bus1, there is 78.9kW of power that can be used to supply an increase in load demand. If Generator 2 fails, then the network can be reconfigured such that some of the 78.9kW of available power is directed to the loads on the second side that need it.

In the HyPSO tool, to determine the power available at a routing node (bus bar), “Determine Available Power” must be enabled in the “Routing Nodes” tab, as shown in Figure 20. Available power will be determined for the routing nodes at the output of each generator, such that the available power for each generator will be obtained.

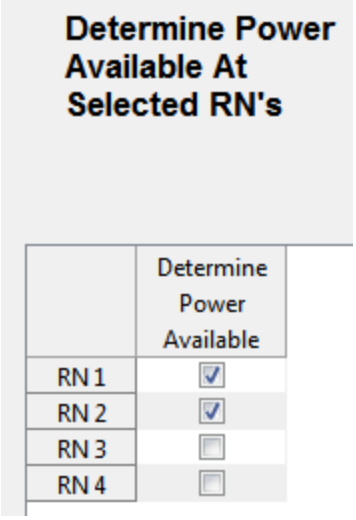


Figure 20: Enable "Determine Available Power" in "Routing Node" Tab

RECONFIGURATION EVENT

A reconfiguration event is demonstrated using the large test architecture. Generator 2 is set to fail at a simulation time of 200 seconds. The failure can be seen in the Machine Power and Efficiency curve of Figure 21: Machine 2 (generator) drops to an output power of 0kW at 200s.

A reconfiguration event also occurs at 200s. The contactor of Reconfigurable Path 20 closes and current begins to flow from Routing Node 3 to Routing Node 4. Thus, Generator 1 is supplying the loads that Generator 2 can no longer supply. We can see from the graph that, indeed, Generator 2 is now supplying double its original load since each side of the network has identical loads.

Finally, at 400s the second possible reconfiguration configuration is demonstrated. Reconfiguration Path 20 opens so that Generator 1 is no longer supplying power to the second path, and Reconfiguration Path 13 closes such

that Generator 4 is now supplying power to the second path. This is a significant change because now Engine 2 is supplying electrical power to three paths and Engine 1 is supplying one path, whereas from 0-400s both engines were supplying two paths each. From the graph we can see that now Machine 4, (Generator 4) is supplying double the output power.

Perhaps a more significant graph is the Available Power at the routing nodes. The available power is determined by the different of the power being generated upstream to that of the power drawn by loads downstream. So for Routing Node 3, the available power is the difference between the power being generated by Generator 1, and the power being consumed by Load 1 and Machine 5 (TRU 5).

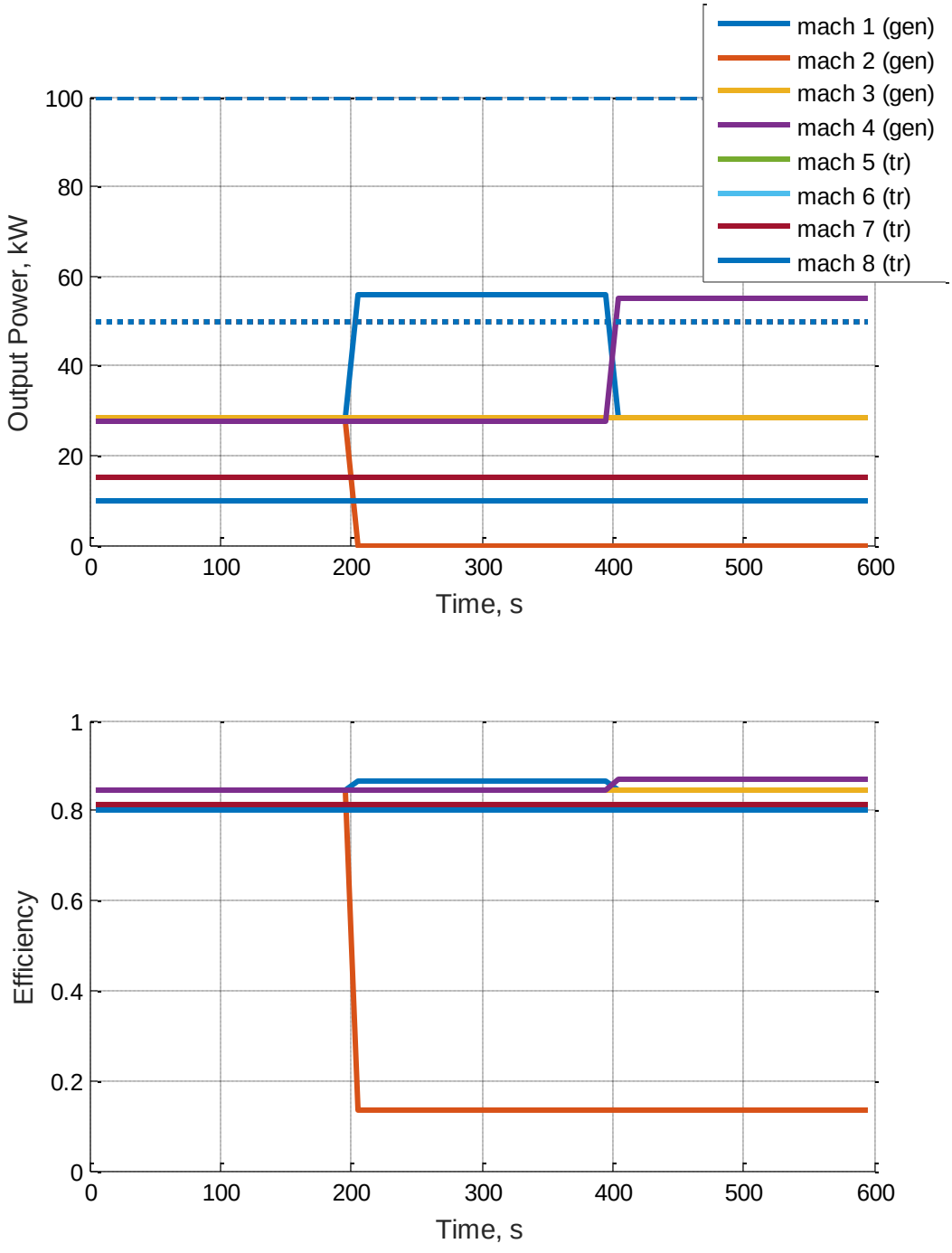


Figure 21: Machine Power and Efficiency

The tool has several outputs, of which these graphs are just a few. An aircraft designer may be most interested in parameters such as Fuel Flow in

Figure 23 or Available Power in Figure 22. The analysis performed hereafter will mostly be concerned with Available Power. As previously discussed, analyzing available power will assist aircraft designers in sizing problems. Markov analysis is first used to determine which states are most likely to occur. The Tool can then be used to simulate a reconfiguration event to reach that state. In that state, data on the available power may be collected. Using the state with the worst case available power, the generator can be downsized to minimize the excess Available Power.

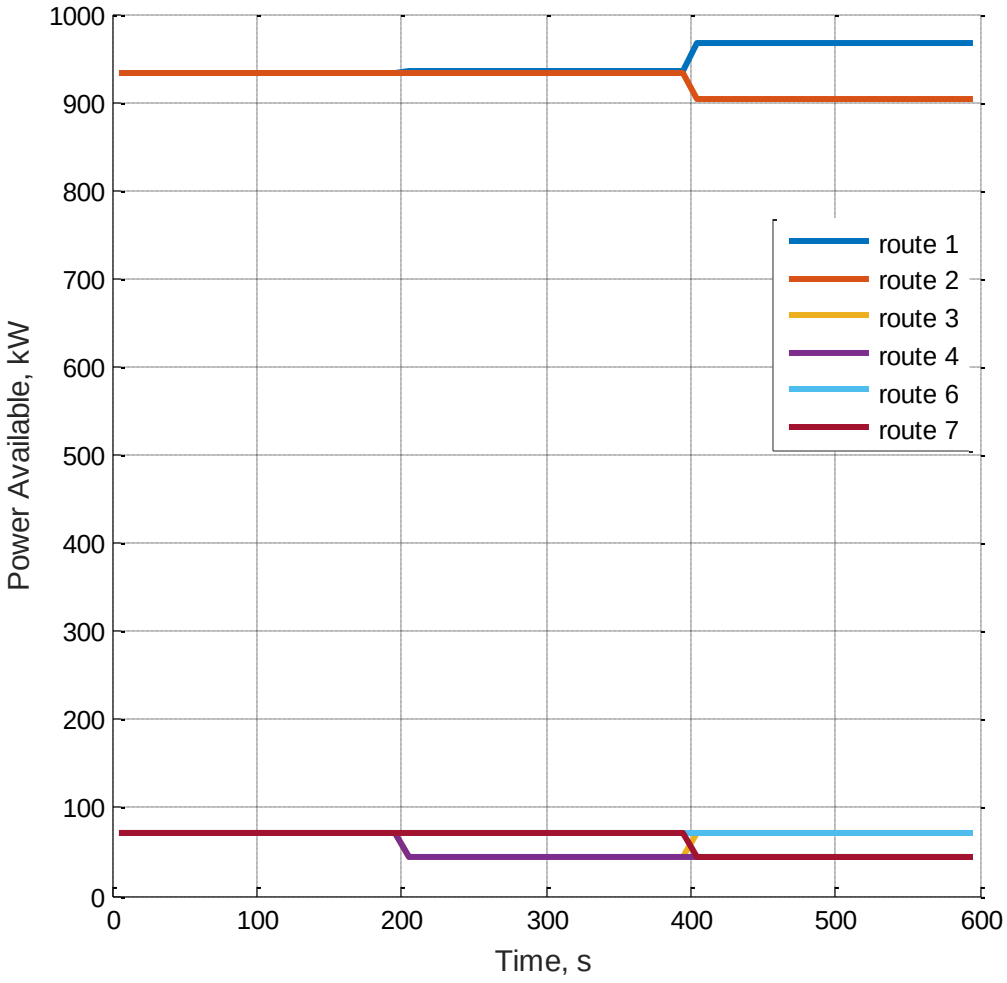


Figure 22: Available Power at Routing Nodes

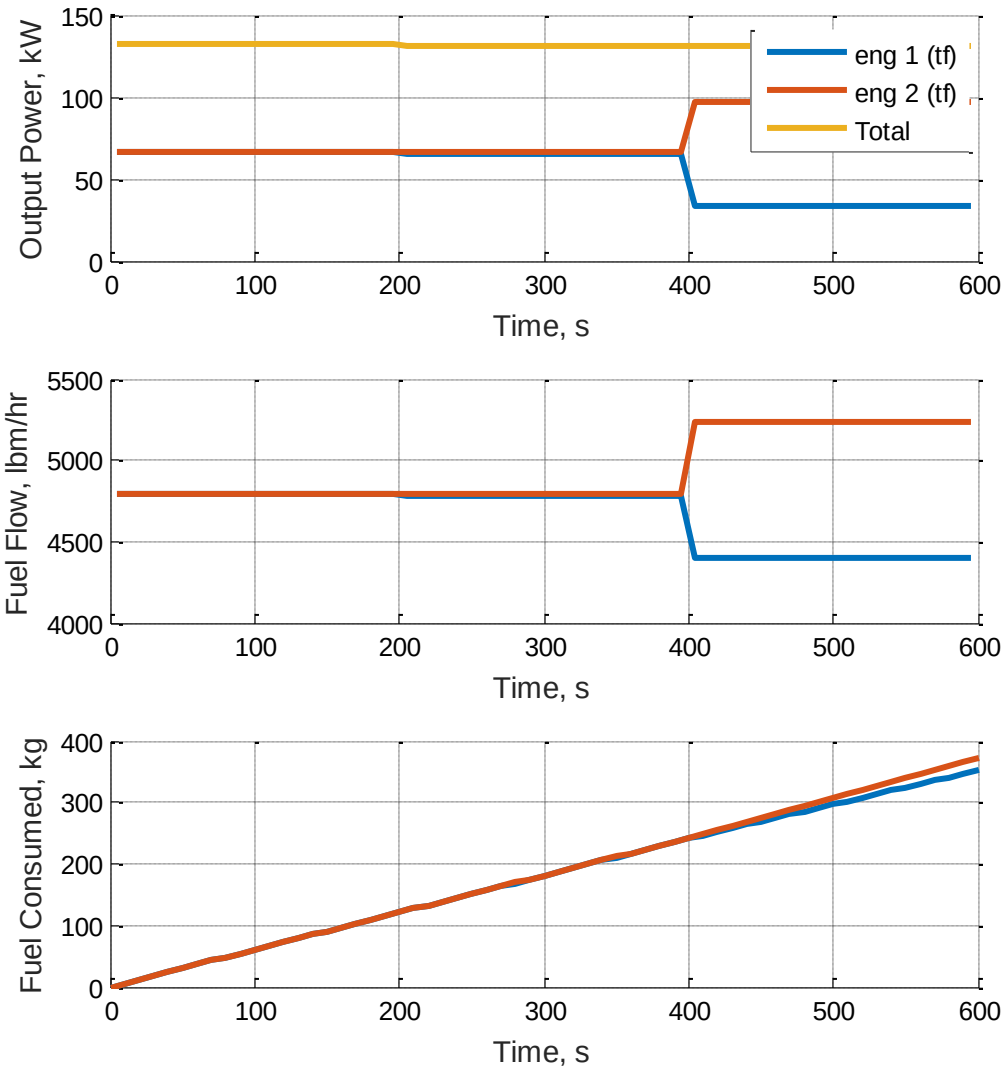


Figure 23: Engine Performance

HyPSO ADD-ON DESCRIPTION

Before Markov analysis may be performed on the system, the system needs to be built and described in HyPSO. It begins in yEd, with the custom palette previously described. See Appendix A and Appendix B for the yEd

architectures with HyPSO numbering used for the analysis in this project. These architectures are imported into HyPSO, and the Tool automatically parses and creates the simulation environment. The only manual input required is the machine efficiency curves and the enabling of Available Power calculations discussed in the HyPSO Setup section.

The procedure for performing Markov analysis will be performed in the following steps:

1. Generate state space with state probabilities
2. Display in MATLAB data structure
3. Identify trivial and collapsible states
4. Collapse state space by summing state probabilities
5. Determine available power and load availability and attach to collapsed states
6. Generate data visualizations

These steps will now be discussed in detail:

1. GENERATE STATE SPACE

Using the Symbolic Math Toolbox provided by MATLAB, the system of differential equations may be solved. This is achieved in the stProb.m function (see Appendix E.3). Given the system state space structure and the number of flight hours to simulate over, this function will computer the probability of each state being reached over the number of flight hours. Flight hours can also be

input as an array (a MATLAB row vector) to simulate over several ranges of flight hours at the same time.

2. DISPLAY IN MATLAB DATA STRUCTURE

Next, in MATLAB, a data structure is used to store and organize the system description. The data structure is built in buildSys.m (see Appendix E.2). The data structure is formatted as a tree. The top level of the tree is the nominal system state with no failures. It contains the probability that no failures will occur over the flight hours The next level of the tree is the state of the system after a single failure. Each of these states contains an equation representing the probability that the state will occur after a number of flight hours. The three level of the tree is the state of the system after three failures. At the moment, this data structure only contains three levels. However, it would be genericized to include more levels, each new level representing a new failure.

3. IDENTIFY TRIVIAL AND COLLAPSIBLE STATES

The rules for collapsing states are discussed in the Markov Analysis section. A MATLAB script is written to execute these rules in elimTrivial.m seen in Appendix E.5.

4. COLLAPSE STATE SPACE BY SUMMING PROBABILITIES

The collapsible states identified in step 3 are then combined in the MATLAB data structure. To combine the states, the probabilities of being in the

collapsible states are summed. The MATLAB script that performs this summing is aggregate.m seen in Appendix E.6.

5. DETERMINE AVAILABLE POWER AND LOAD AVAILABILITY

The next step is to extract the Available Power and load availability data from the HyPSO simulation and attach it to the data structure. First, the HyPSO simulation should be run to generate the required data. The failure and reconfiguration events are auto-populated in the Tool using the statusize.m script in Appendix E.8. Next, the output of the simulator is parsed and attached to the data structure built in steps 1-4. This action is performed by getPwrAvail.m seen in Appendix E.7. Since HyPSO is also developed in MATLAB, it is a simple task to extract the output of the simulator and attach it to the data structure such that the relationship between the state space and available power may be determined.

6. GENERATE DATA VISUALIZATIONS

Now that the data structure has been built by creating the state space, collapsing the sate space, running a HyPSO simulation, and parsing the output data, we need a method to view the data in a comprehendible way. MATLAB has strong graphing tools that were employed for this purpose. Several data visualizations were developed and shall be discussed in the next section. The data visualization MATLAB code is seen in Appendix E.9 through Appendix E.14.

V. EXAMPLE ANALYTICAL APPROACH

SMALL TEST ARCHITECTURE

This method was tested on the small test architecture of Figure 8. The test was performed with the failure rates given in Table 4:

Table 4: Failure Rates for Small Test Architecture Test # 1

Type	MACH1 MACH2	MACH3 MACH4	BUS1 BUS2	BUS3 BUS4
Name	GEN1 GEN2	TRU1 TRU2	AC1 AC2	DC1 DC2
Fail Rate	7E-4	8E-5	1E-6	1E-6

After aggregation but before eliminating trivial states, a diagram of the state space is generated, as in Figure 24. This figure shows the probabilities that each state in the state space will occur after 10 flight hours. It is color coded by rate of occurrence. Note that the “Probability of Failure” axis is on a logarithmic scale. From this diagram it is clear that if a particular aircraft has a 10 hour route, either Machine 1 or Machine 2 (Generators 1 or 2) will fail frequently. In a slightly more frightening case, it is reasonably probable that both Machines 1 and 2 will fail at the same time. It is also reasonably probable that either TRU 1 or TRU 2 (Machines 3 or 4) will fail. All other states are remote or for all practical purposes will never occur. This diagram is useful for seeing which states should be focused on and planned for. If designers know that it is reasonably probable that Generators 1 and 2 can fail at the same time, then a mitigation plan can be developed.

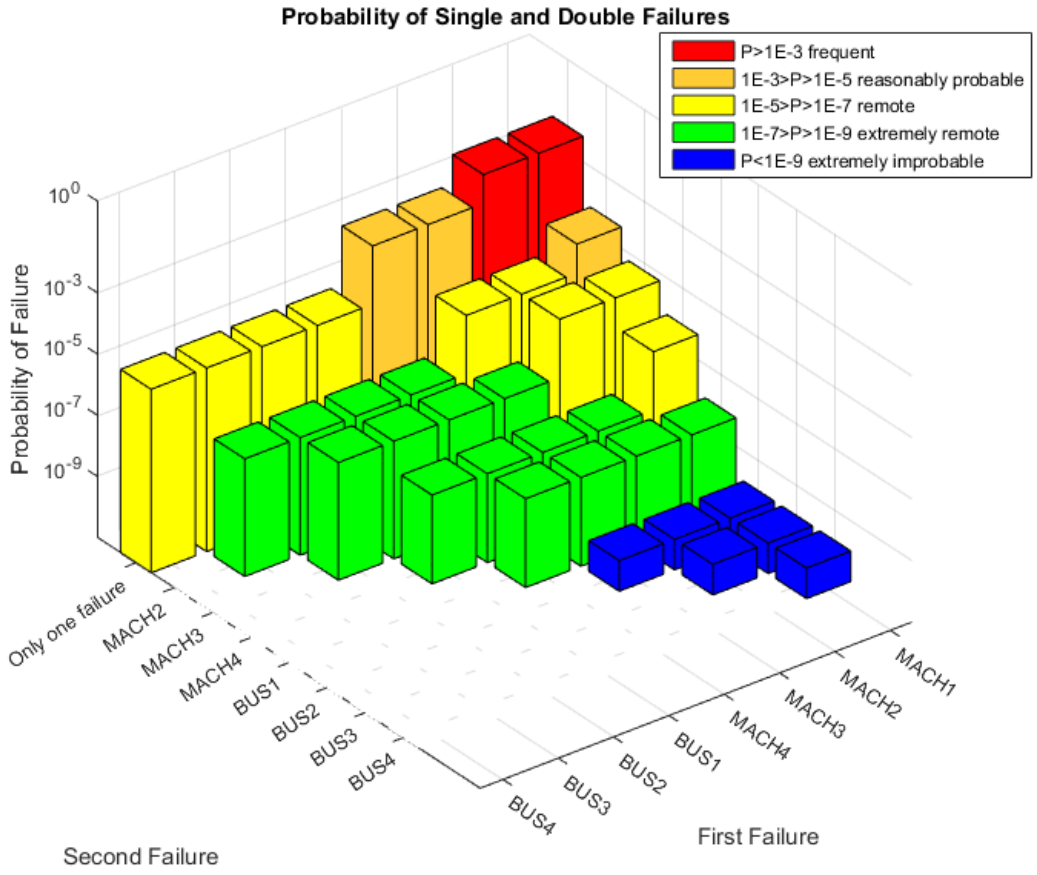


Figure 24: Aggregated State Space for Small Test Architecture (10 flight hours)

In Figure 25, the probability of at least one load not being serviced with power is shown over time. After 10 flight hours, there is a 9*10^-5 chance, or about 1 in 11,000 chance that an electrical load will be without power. For a 10 hour route, if the failure rates remain constant than at least one flight in 11,000 will see a load failure.

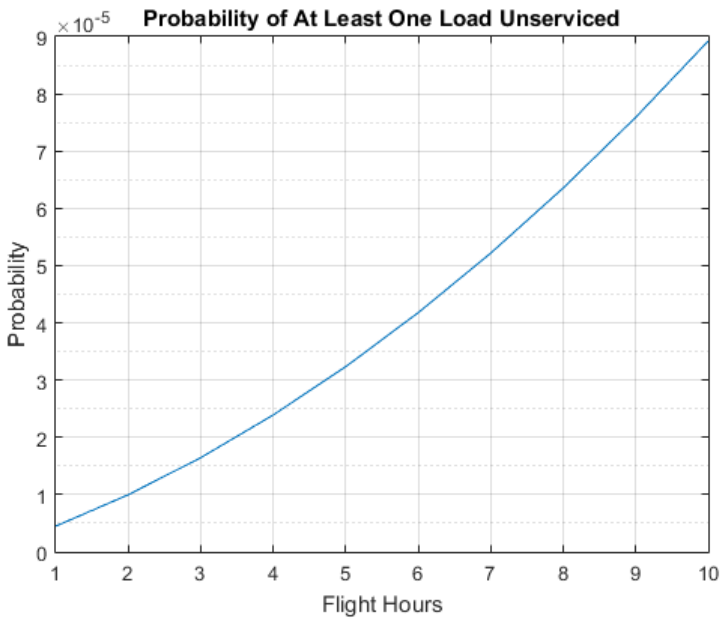


Figure 25: At Least One Unserviced Load in Small Test Architecture

Figure 26 displays probabilities over a much longer time period. This view could be useful for scheduling maintenance on the aircraft generators. Since Figure 24 showed that we will frequently encounter generator failures, we can see exactly how much time it will take to almost guarantee a generator failure. Note that the Probability axis is on a logarithmic scale. The probability that there will be absolutely no failure on the aircraft decreases logarithmically. At about 500 flight hours is a critical point: this is when there is a higher probability that there will be at least 1 generator failure becomes greater than no failure occurring at all. Perhaps at this point generator maintenance should be scheduled. Also notice how the chance of only one generator failure actually decreases in time after an inflection point at about 1000 flight hours. This is because it is becoming more likely that there will be additional failures, so the single failure state would transition to a multiple failure state.

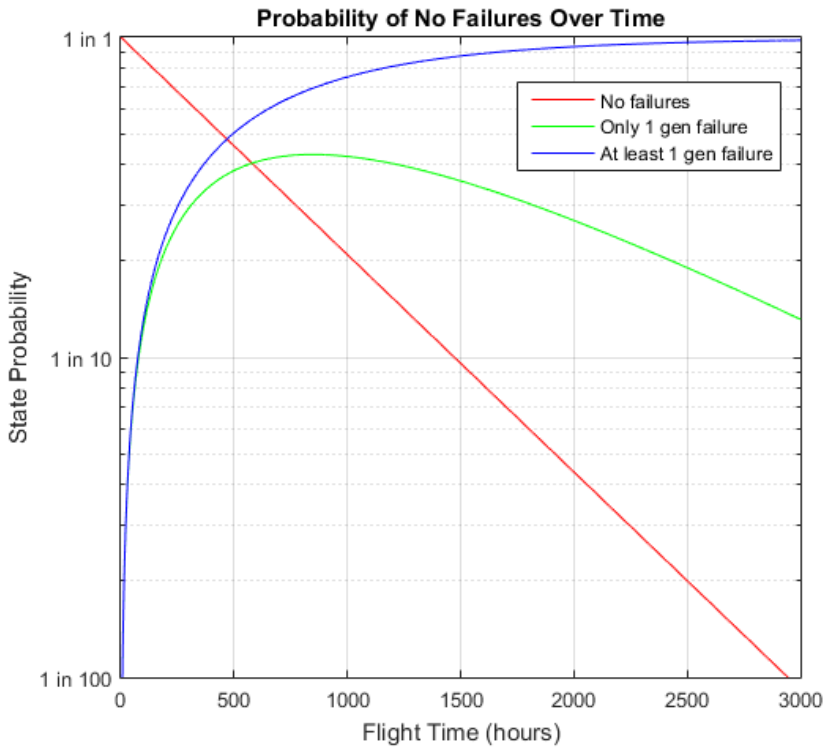


Figure 26: Small Test Architecture, No Fails and Generator Fails

Figure 27 takes into consideration both the load availability and the Available Power at each of the generators for each of the single failure states. For the small test architecture, the only way a single failure can cause a load outage is if the bus bar that the load is attached to fails. According to this hypothetical system, bus bars are resilient systems that don't fail as often as the machines. The low probability of the load outage occurring makes it less significant of a consideration when designing redundancy into the network.

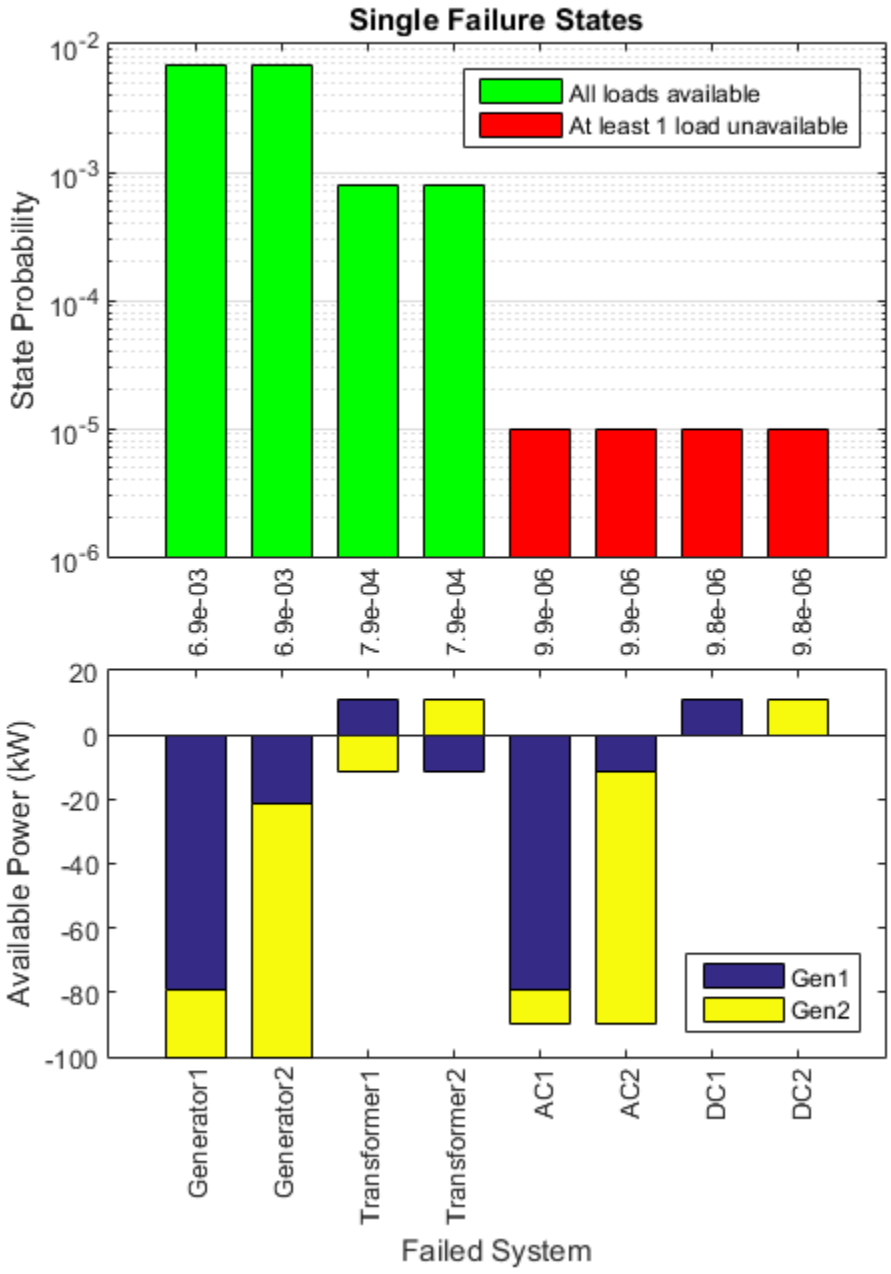


Figure 27: Load Availability and Available Power

for Small Test Architecture (10 hours)

In the second subplot, the available power for each of the generators is displayed. This chart shows the difference from nominal. For example, if Generator 1 usually has 80 kW of available power and it fails, there is now 80kW less available power than the nominal case. Thus, -80kW available for Generator 1, and -20kW available for Generator 2 because it now has to compensate for the loads the Generator 1 used to be supplying. In this example, Generator 1 had a maximum output power of 100kW. After it fails there is a total of 100kW less power available in the network, which is why the available power of Generator 1 and Generator 2 add up to -100kW.

From Figure 27 it is apparent that either Generator 1 or 2 failing causes the highest deficit of available power. When considering just one failure, this would be the "sizing case." The sizing case means that the generators would be sized based on how much available power there was in the network after one of the generators has failed.

It is also important to consider positive available power. This means that there is more available power at the generator now than before the failure. For the DC bus bar failure case, there is about 10kW more available power than before. It can be inferred from this data that the 10kW load attached to the DC bus bar is no longer being serviced with power.

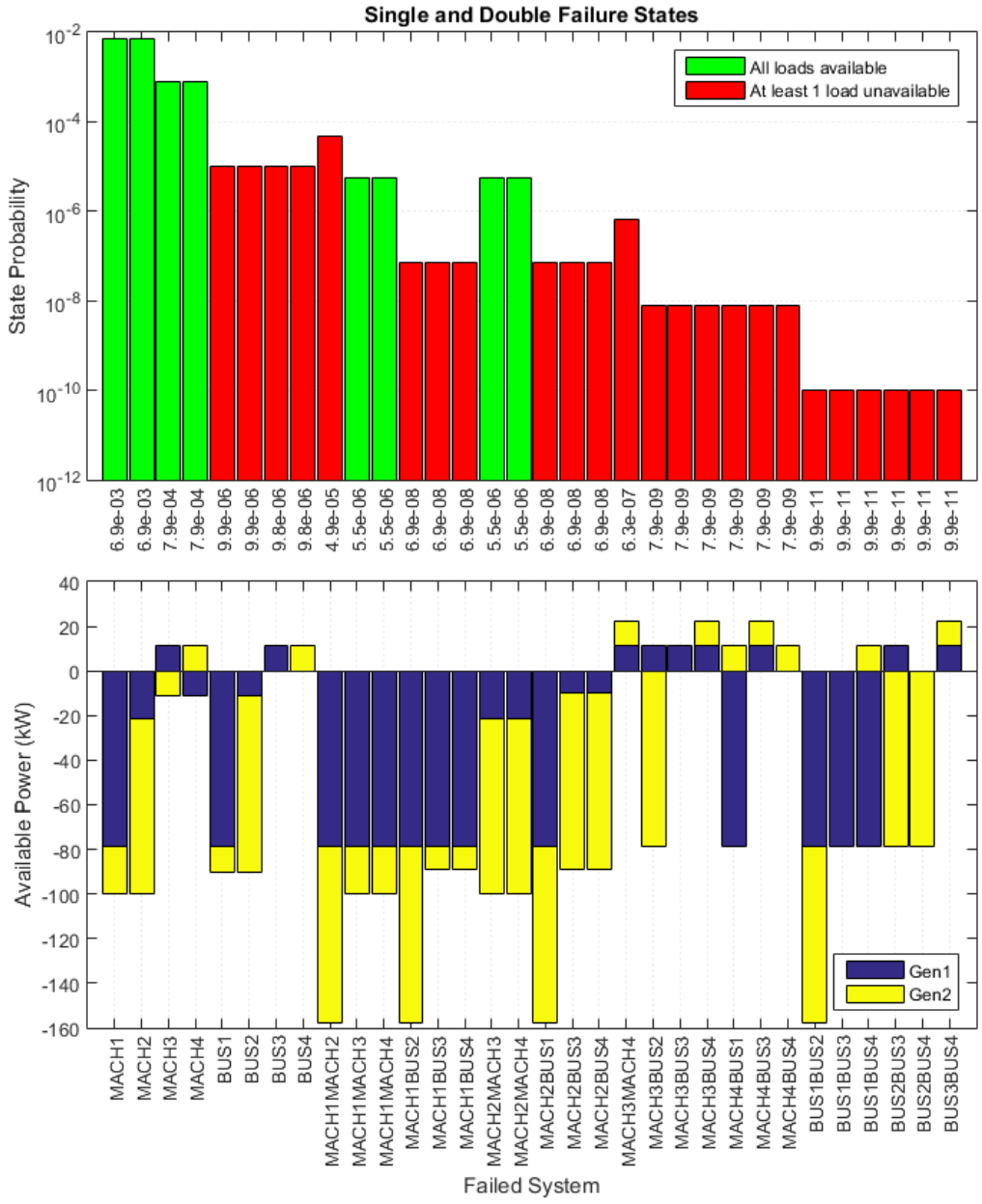


Figure 28: Single and Double Failures,

Load Availability and Available Power (10 hours)

The plot in Figure 28 expands the plot from Figure 27 to include double failure states. Here there is a full view of the architecture. This plot will begin to get cluttered, however, as larger state spaces are considered.

The state space shown in Figure 24 has a symmetric feel to it only because the failure rates provided for the systems are simple examples. Hypothetically, after a round of testing and FMEA, designers could find that DC Bus 2 (Bus 4) is particularly susceptible to failure and has a higher rate than other buses. The state space is regenerated in Figure 29 using the values of Table 5 to provide a more realistic view of what a state space make look like in practice. The advantage of viewing these plots in MATLAB is the ability to rotate the graph and get a full view of the system.

Table 5: Failure Rates for Small Test Architecture Test # 2

Type	MACH1 MACH2	MACH3 MACH4	BUS1 BUS2	BUS3 BUS4
Name	GEN1 GEN2	TRU1 TRU2	AC1 AC2	DC1 DC2
Fail Rate	7E-4 5E-4	1.2E-4 8E-5	6E-6 3E-6	1E-6 1E-3

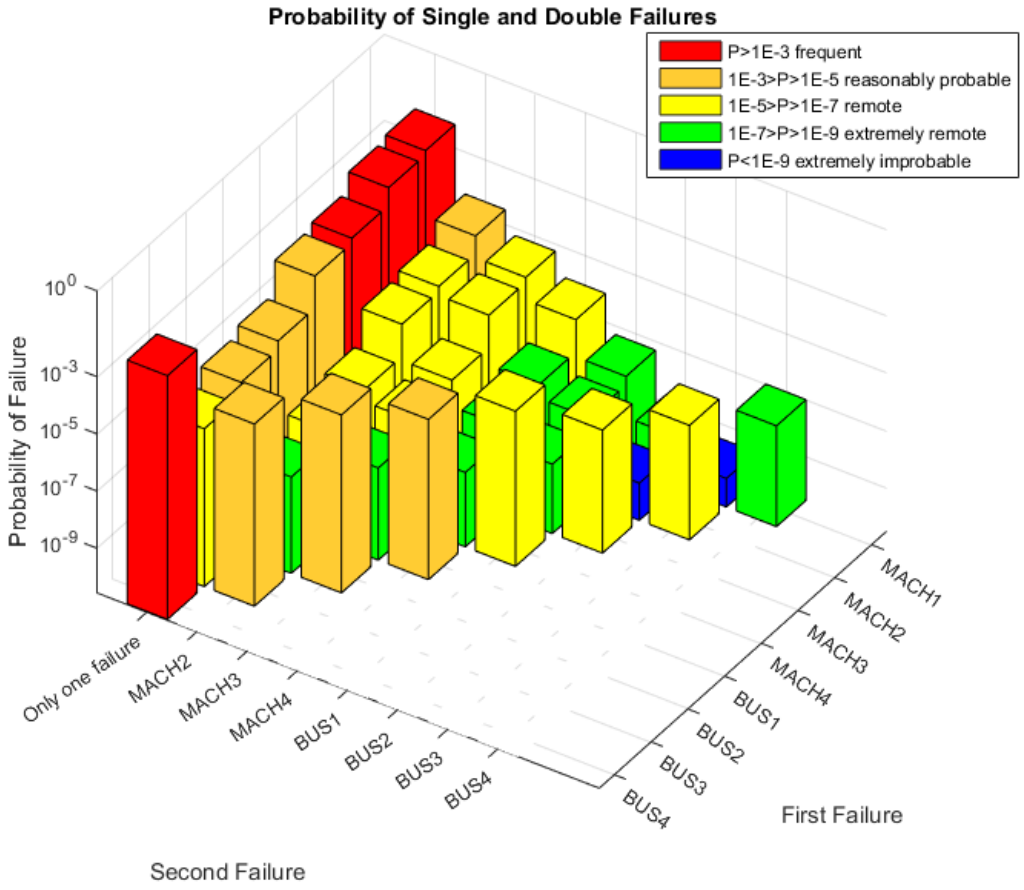


Figure 29: Aggregated State Space #2

FULL TEST ARCHITECTURE

Figure 30 displays the state space for the full test architecture, previously shown in Figure 18. The failure rates used for this run are the same as those used for the small test architecture in Table 4.

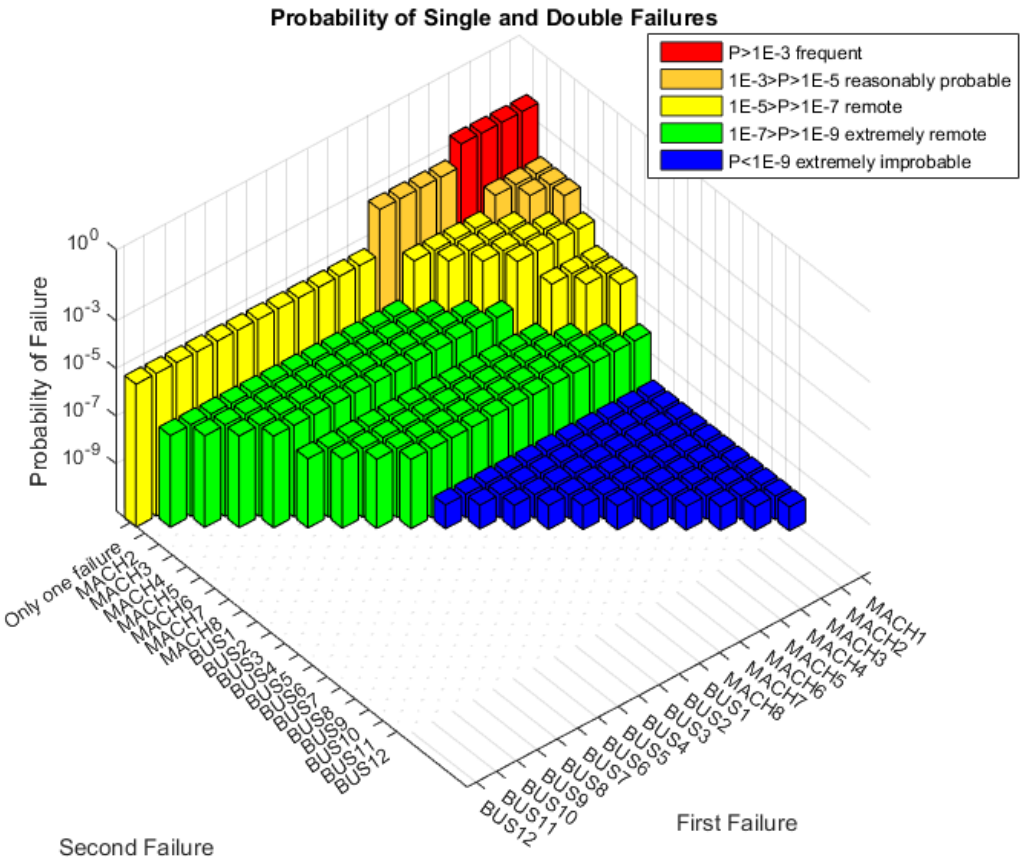


Figure 30: Aggregated State Space for Test Architecture

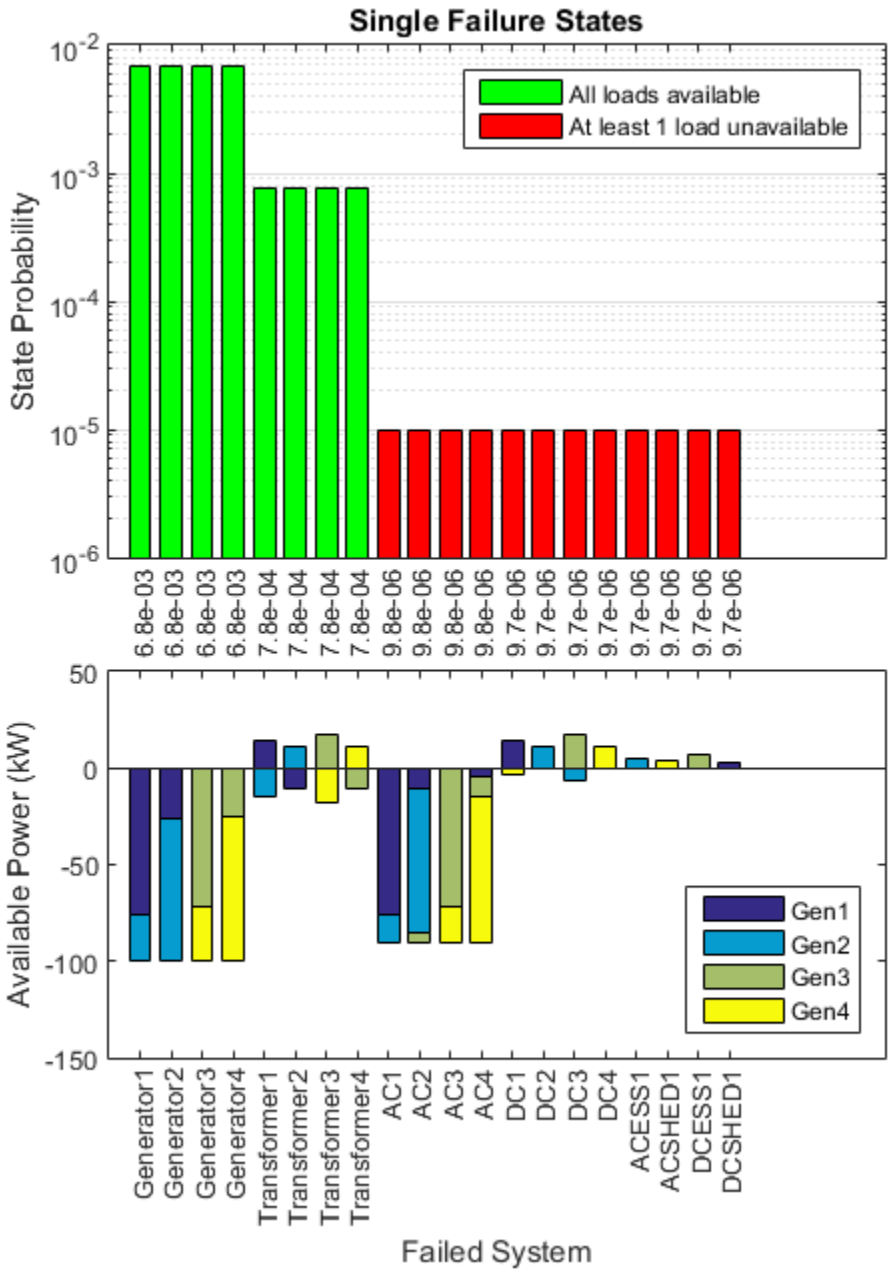


Figure 31: Load Availability and Available Power for Full Test Architecture

In Figure 31, the available power is now divided amongst the four generators, as opposed to 2 in the previous example. There are too many states for the single and double failure plot (analogous to Figure 28) to be useful in this case.

Compared to the smaller architecture, the full test architecture has a higher chance of having at least one load unserviced, as seen in Figure 32. At the end of 10 flight hours, there is a 1.2E-4 probability, or about 1 in 8300, that a load will fail, compared to 1 in 11000 for the smaller architecture. This is due to the fact that there are more ways in which a load can be unserviced. For example, there are now 12 buses instead of 4, any if any one of the buses fail then a load is unserviced. If all of the buses on both architectures are said to fail at the same rate, then the architecture with 3 times as many buses will have a 3 times higher chance of a load failing due to a bus failure.

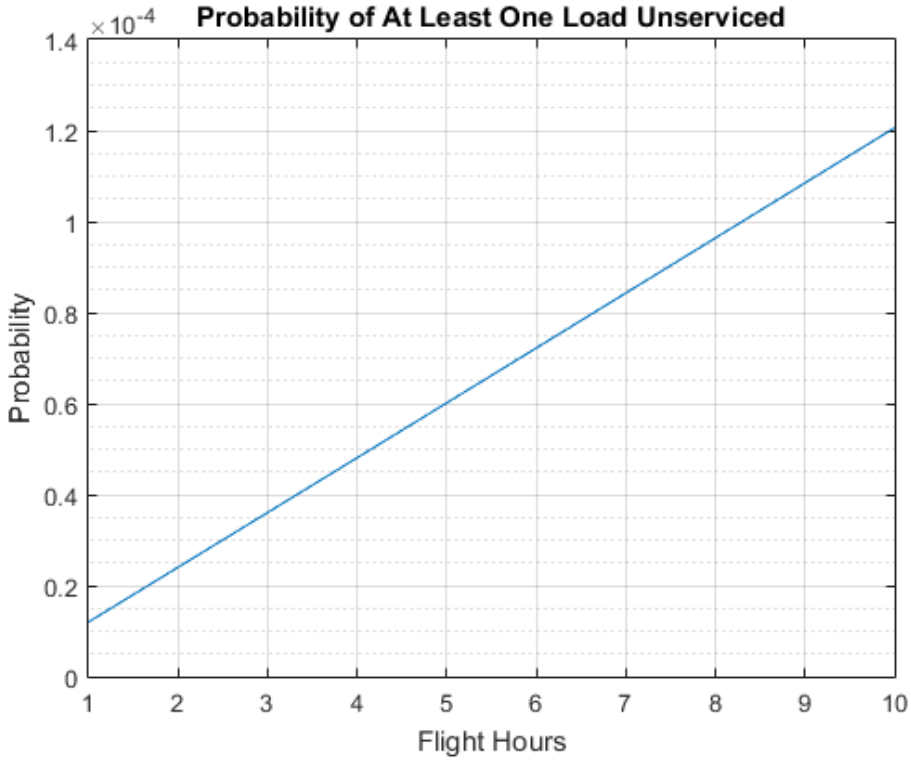


Figure 32: At Least One Unserviced Load in Full Test Architecture

These data visualization tools are useful for comparing two similar architectures to see which one is more advantageous. They can be used as a

rating method to objectively determine which architecture is cheaper, safer, or both.

Electric power architecture designers must strike a balance between operating costs and safety/reliability. Sufficient redundancy must be built into the network to ensure passenger safety. However, too much redundancy will unnecessarily add excess weight to the aircraft, significantly increasing operating costs. Contactors in particular are significant sources of weight. If a contactor could be eliminated from the network without a significant increase in risk, then the aircraft manufacturer could increase its competitiveness in the marketplace by lowering operating costs.

To see if contactors can be eliminated safely, the test architecture of Figure 18 will be modified to remove Reconfigurable Paths 5 and 8. This is equivalent to removing two contactors from the network. The modified architecture is given in Figure 33. The HyPSO numbering matches the reference given in Appendix B.

For this analysis, only the states in which Reconfigurable Paths 5 and 8 are closed are needed. All other data will remain the same. The probability of reaching a state in which these paths would be closed remains the same, since the system components and failure rates do not change. The main variable that will change is the available power. Figure 34 can be directly compared to Figure 31. By running multiple simulations on slightly modified architectures, designers can determine which architecture is superior.

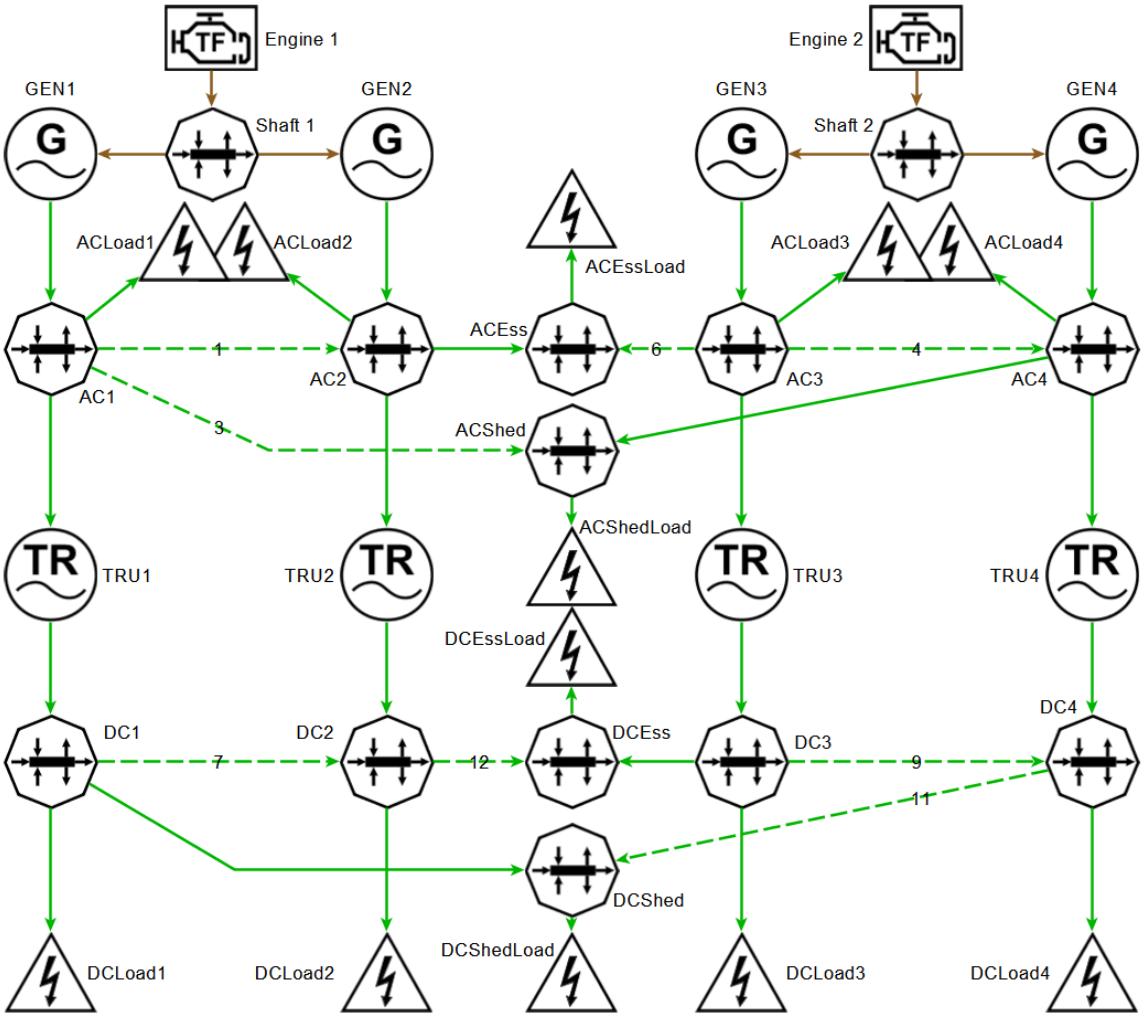


Figure 33: Modified Full Test Architecture

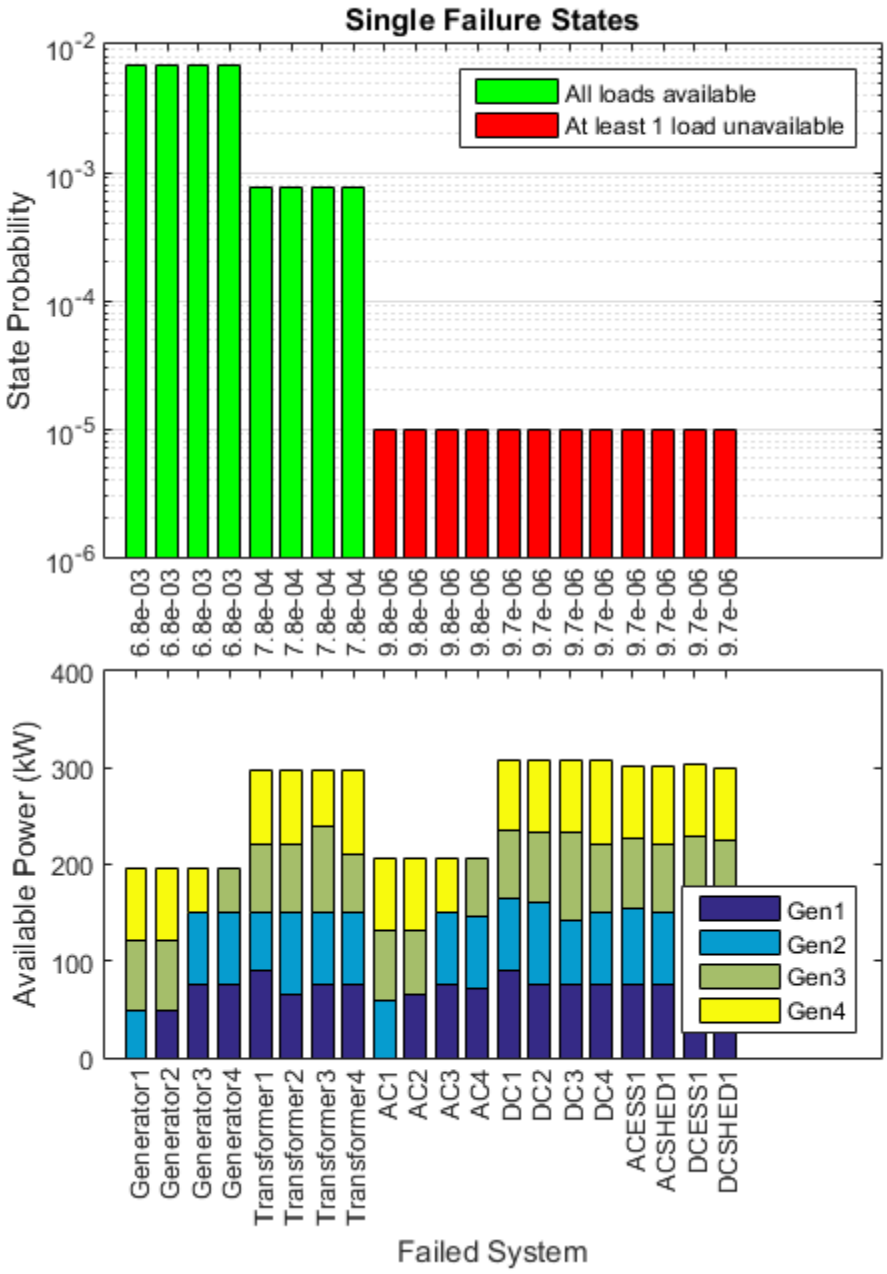


Figure 34: Available Loads and Power for Modified Full Test Architecture

VI. CONCLUSION

As more research and development is performed in the aerospace industry in the field of More Electric Aircraft, better tools are needed to analyze increasingly complex aircraft electric power systems. Airbus developed the HyPSO tool to simulate hybrid power systems. There are many outputs to the Tool,, one of which is the Available Power at bus bars. Available power is an important parameter in the sizing of electric power systems. The Tool is capable of simulating failures in various power systems, and reconfiguring the power system as would be done in an actual failure scenario. This feature was employed heavily in the analysis performed in this project.

To predict which types of failures would be most common, and thus of most concern and deserving of the most attention, Markov analysis was performed. A state space was created assuming a few failure modes. Transitioning between states occurs when a failure occurs. The probability of being in each state was calculated. An add-on to the HyPSO tool was created in MATLAB to perform the calculations involved in Markov analysis. The add-on is structured around a tree data structure which contains each of the states in the state space. MATLAB solves the system of ordinary differential equations – one equation representing each state in the state space. The solution to the set of equations is the probability that a state will be reached in a certain amount of time. The data structure stores the equation associated with each state with respect to time t, so several time values may be input, generating probability over time. The add-on generates failure and reconfiguration events and

auto-populates the events into HyPSO. Once a HyPSO simulation is run, data associated with each failure and reconfiguration event is parsed and stored in the data structure in the state associated with the event.

In an actual analysis of a real aircraft power system, the architecture input into HyPSO is likely to be much more complex than the architectures analyzed in this project. In addition, several more failure modes may be programmed into the add-on. In this case, there may be massive amounts of data generated and stored into the data structure. To assist in data analysis, several data visualizations were created in MATLAB. These data visualizations help the user to draw connections between probable failure states and the Available Power in these states. Two use cases were performed, where the data visualizations were employed to analyze a small and a full test architecture. The data visualizations were helpful in predicting when a critical failure would occur, or when an electrical load would be without power. They can also be used for other purposes, such as scheduling generator maintenance. Finally, plots of Available Power were developed to be used for power system sizing.

Available Power is the primary parameter involved in sizing of aircraft power systems. Using the data structure created as an add-on to HyPSO tool, aircraft designers can determine which failure states are most likely to occur. Using the existing output of HyPSO, which is parsed into the data structure by the add-on, designers may determine the Available Power at the generator in these likely-to-occur states. They may then size the generators appropriately so that as many electrical loads as possible are serviced during a failure. If a

generator may be down-sized, weight is saved in the aircraft, thereby reducing the amount of fuel during flight. When fuel is saved, operating costs are decreased, and the aircraft manufacturer can gain a competitive advantage in the marketplace.

FURTHER RESEARCH

HyPSO could benefit from several changes and additions to the software. First, a better mechanism to cause the failure of bus bars (routing nodes) should be implemented. For this project, the main input power path was disconnected (set to zero) in order to fail the routing nodes. This method could cause errors if the routing node is connected to another source of power, especially another routing node. The automatic failure feature built in this add-on does not check for other input power paths to the routing node. In reality, if a bus bar fails, nothing can supply or be supplied by the bus bar. It is not sufficient to say that the input power path is failed.

Secondly, the Tool could benefit from a “load availability” feature. Instead of just declaring that the simulation constraints have been violated, report the number of loads that are unable to be supplied with power, or the magnitude of the power shortage. It appears that there is no correlation to the values in opt.constrVio and the amount of the actual power shortage. Thus, only a binary output can be inferred: if there is a constraint violation, then there must be at least one load that is not properly supplied with power, and if there is no constraint violation, all loads are supplied with power.

The add-on developed in this project could be expanded to extend its functionality. The simplest and most effective expansion would be the addition of new data visualizations. The user may be interested in different results other than those presented here. With the provided data structure, simple manipulations can assist the user in better understanding the system under test. The add-on may be expanded to include three or more failure states.

The Markov chain presented here is non-recoverable. When a state is entered, it is not possible to return to the original state. In reality, systems are repairable. There is a chance that the system can return to a previous state. This return is described by a transition rate, same as the progression transition rates used in this analysis. Repairable systems as described in ARP4761 [9] could be implemented in this add-on. Since HyPSO mostly centers around mission-based simulations, e.g. a single flight at a time, this project did not cover repairable systems, which is more of an aircraft lifecycle or up-time concern.

This method could be useful if built on the MATLAB Simulink platform. A Simulink block could describe a state, then connections between the blocks would be described by transition rates. In this way, a state space could be built in as a block diagram, with the back-end code hidden to the user. This would greatly assist in the usability of the add-on and in the comprehension of the complex state space of the system.

REFERENCES

[1] Shappell, S., Detwiler, C., Holcomb, K., Hackworth, C., Boquet, A. and Wiegmann, D. (2007). Human Error and Commercial Aviation Accidents: An Analysis Using the Human Factors Analysis and Classification System. Human Factors, 49: 227-242.

[2] “Accident Statistics.” www.planecrashinfo.com. Accessed 03 June 2015.

[3] Mouawad, Jad; Drew, Christopher, “Airline Industry at Its Safest Since the Dawn of the Jet Age.” The New York Times, 11 February 2013. Accessed 09 September 2015. http://nyti.ms/18oXqeS

[4] “Death Rate per Year.” The Bureau of Aircraft Accidents Archives. Geneva, Switzerland. www.baaa-acro.com/

[5] “Motor Vehicle Accidents—Number and Deaths: 1990 to 2009.” U.S. National Highway Traffic Safety Administration, Traffic Safety Facts, annual; and unpublished data. http://www.census.gov/compendia/statab/2012/tables/12s1103.pdf

[6] Brombach, J.; Schroter, T.; Lucken, A.; Schulz, D., "Optimized cabin power supply with a +/− 270 V DC grid on a modern aircraft," Compatibility and Power Electronics (CPE), 2011 7th International Conference-Workshop , vol., no., pp.425,428, 1-3 June 2011.

[7] Wheeler, Pat. “The More Electric Aircraft: Why Aerospace Needs Power Electronics.” University of Nottingham, Nottingham, UK

[8] Ahmed Abdel-Hafez (2012). Power Generation and Distribution System for a More Electric Aircraft - A Review, Recent Advances in Aircraft

Technology, Dr. Ramesh Agarwal (Ed.), ISBN: 978-953-51-0150-5, InTech, Available from: http://www.intechopen.com/books/recent-advances-in-aircraft-technology/more-electric-aircraft

[9] Sinnet, Mike. “787 No-Bleed Systems: Saving Fuel and enhancing operational efficiencies,” Boeing. Aero Quarterly, 4th quarter 2007.

[10] Rafal, K., et al. "Hybridization of an aircraft emergency electrical network: Experimentation and benefits validation." Vehicle Power and Propulsion Conference (VPPC), 2010 IEEE. IEEE, 2010.

[11] ARP4761 “Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment” Aerospace Recommended Practice, Society of Automotive Engineers. Issued Dec 01 1996, S-18, Aircraft And Sys Dev And Safety Assessment Committee.

[12] Michalko, Rodney G. "Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle." U.S. Patent No. 7,439,634. 21 Oct. 2008.

[13] Yue, Edwin, Jean-Luc Derouineau, Wayne Y. Pearson. “Paralleled HVDC bus electrical power system architecture.” U.S. Patent No. 7,936,086. 3 May 2011.

[14] PS462 Datasheet “200 Amp Transformer/Rectifier Unit” L-3 Communications, Marine & Power Systems. http://www.l-3mps.com/products/products-details.aspx?id=220

[15] Moir Ian, Seabridge Allan. Aircraft Systems: Mechanical, Electrical and Avionics Subsystems Integration 3rd, 2008; 9780470059968

BIBLIOGRAPHY

Boudjit, K. ; Korzet, W. ; Nacer, A. ; Moulai, H. ; Larbes, C. “DSP use in distribution power grids - A new approach for high impedance ARC faults detection” Electrical Systems for Aircraft, Railway and Ship Propulsion (ESARS), 2012

Brombach, J.; Jordan, M.; Grumm, F.; Schulz, D. "Converter topology analysis for aircraft application", Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), 2012 International Symposium on, On page(s): 446 – 451

Brombach, J.; Lücken, A.; Nya, B.; Johannsen, M.; Schulz, D. "Comparison of different electrical HVDC-architectures for aircraft application", Electrical Systems for Aircraft, Railway and Ship Propulsion (ESARS), 2012, On page(s): 1 – 6

Brombach, J; Schröter, T.; Lücken, A.; Schulz, D. "Optimizing the Weight of an Aircraft Power Supply System through a +/-270 VDC Main Voltage", in PRZEGLĄD ELEKTROTECHNICZNY (Electrical Review), PL ISSN 0033-2097, R. 88 NR 1a/2012, pp. 47-50

Dolan, Dale, Taufik. “Advanced Power Electronics” EE411 Course Booklet, 2011.

Dolan, Dale, Taufik. “Introduction to Power Electronics” EE410 Course Booklet, 2011.

Eisenhauer, Mark Paul, and Gary Bowman. "Redundant electrical DC power system for aircraft." U.S. Patent No. 6,344,700. 5 Feb. 2002.

La, Jae-Du, et al. "An adaptive control of the bidirectional DC/DC converter with the capacitive energy storage in the more and all-electric aircraft systems."Electrical Machines and Systems (ICEMS), 2010 International Conference on. IEEE, 2010.

Maldonado, Miguel A., and George J. Korba. "Power management and distribution system for a more-electric aircraft (MADMEL)." Aerospace and Electronic Systems Magazine, IEEE 14.12 (1999): 3-8.

Michalko, Rodney G. "Electrical starting, generation, conversion and distribution system architecture for a more electric vehicle." U.S. Patent No. 7,439,634. 21 Oct. 2008.

Montgomery, R. and Galloway, S., "An Optimisation Based Design Approach for Aircraft Electrical Power Systems," SAE Int. J. Aerosp. 7(1):44-52, 2014, doi:10.4271/2014-01-2121.

Norman, P. J., S. J. Galloway, and J. R. McDonald. "Simulating electrical faults within future aircraft networks." Aerospace and Electronic Systems, IEEE Transactions on 44.1 (2008): 99-110.

Phatiphat Thounthong, Stephane Raël, Bernard Davat, Energy management of fuel cell/battery/supercapacitor hybrid power source for vehicle applications, Journal of Power Sources, Volume 193, Issue 1, 1 August 2009, Pages 376-385, ISSN 0378-7753, http://www.sciencedirect.com/science/article/pii/S0378775308024981.

Rakhra, P., et al. "Toward optimising energy storage response during network faulted conditions within an aircraft electrical power system." Electrical

Systems for Aircraft, Railway and Ship Propulsion (ESARS), 2012. IEEE, 2012.

Reddy, V. V. K., and M. Sydulu. "A heuristic-expert based approach for reconfiguration of distribution systems." Power Engineering Society General Meeting, 2007. IEEE. IEEE, 2007.

Roboam, Xavier. "New trends and challenges of electrical networks embedded in “more electrical aircraft”." Industrial Electronics (ISIE), 2011 IEEE International Symposium on. IEEE, 2011.

Rosero, J. A., et al. "Moving towards a more electric aircraft." Aerospace and Electronic Systems Magazine, IEEE 22.3 (2007): 3-9.

Taliaferro, James Bryan. "Power distribution expert system." U.S. Patent No. 7,369,921. 6 May 2008.

Terörde, M.; Lücken, A; Schulz, D. “Weight saving in the electrical distribution systems of aircraft using innovative concepts” International Journal of Energy Research, Volume 38, Issue 8, pages 1075–1082, 25 June 2014

Volp, Jeffrey A. "Fault-tolerant power distribution system." U.S. Patent No. 4,659,942. 21 Apr. 1987.

Wu, Jian Jun, Wei Wan, and Peng Wu. "Design of Aircraft Electrical Load Management Center." Advanced Materials Research 268 (2011): 546-551.

Xu, Huan. “Design, Specification, and Synthesis of Aircraft Electric Power Systems Control Logic.” Dissertation, California Institute of Technology. Pasadena, California, May 31, 2013.

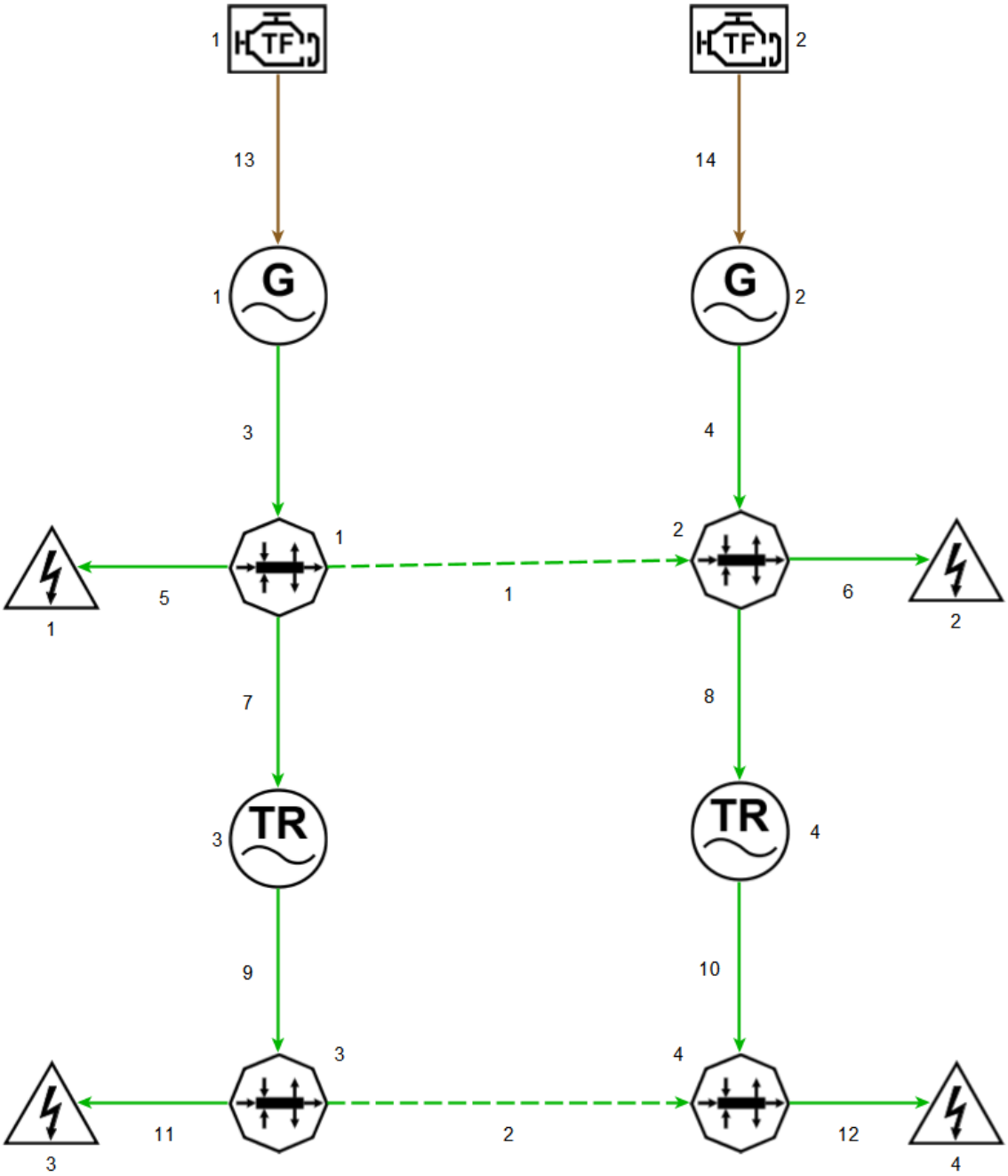
Yang Shanshui; Lexuan Meng; Ligang Ruan; Jian Zhao; Li Wang, "Modeling and simulation of aircraft automatic power distribution system," Electrical Systems for Aircraft, Railway and Ship Propulsion (ESARS), 2012 , vol., no., pp.1,4, 16-18 Oct. 2012

Yu, Helen. EE513 Modern Control Systems course materials.

APPENDICES

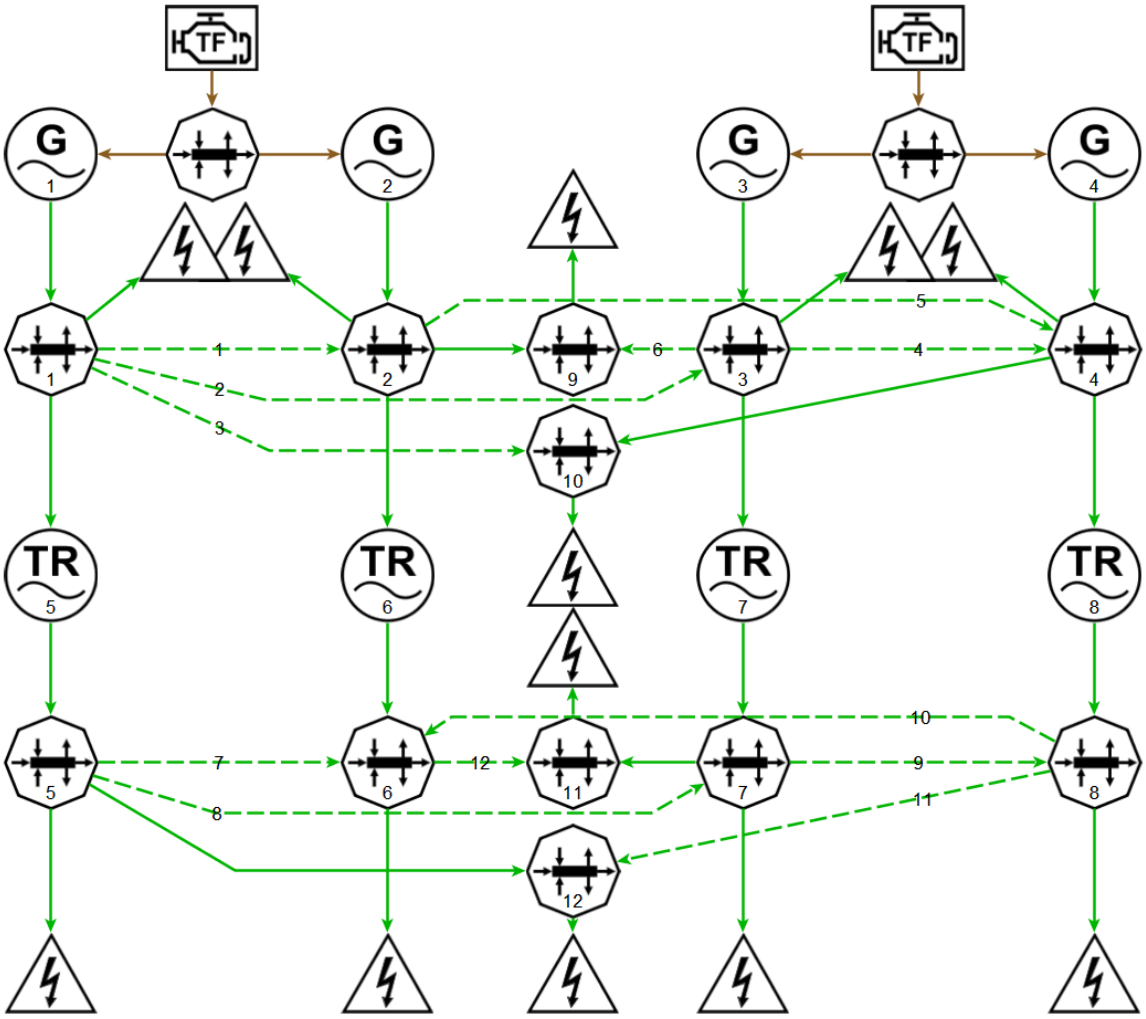
APPENDIX A - SMALL TEST ARCHITECTURE

A.1 - HyPSO NUMBERING

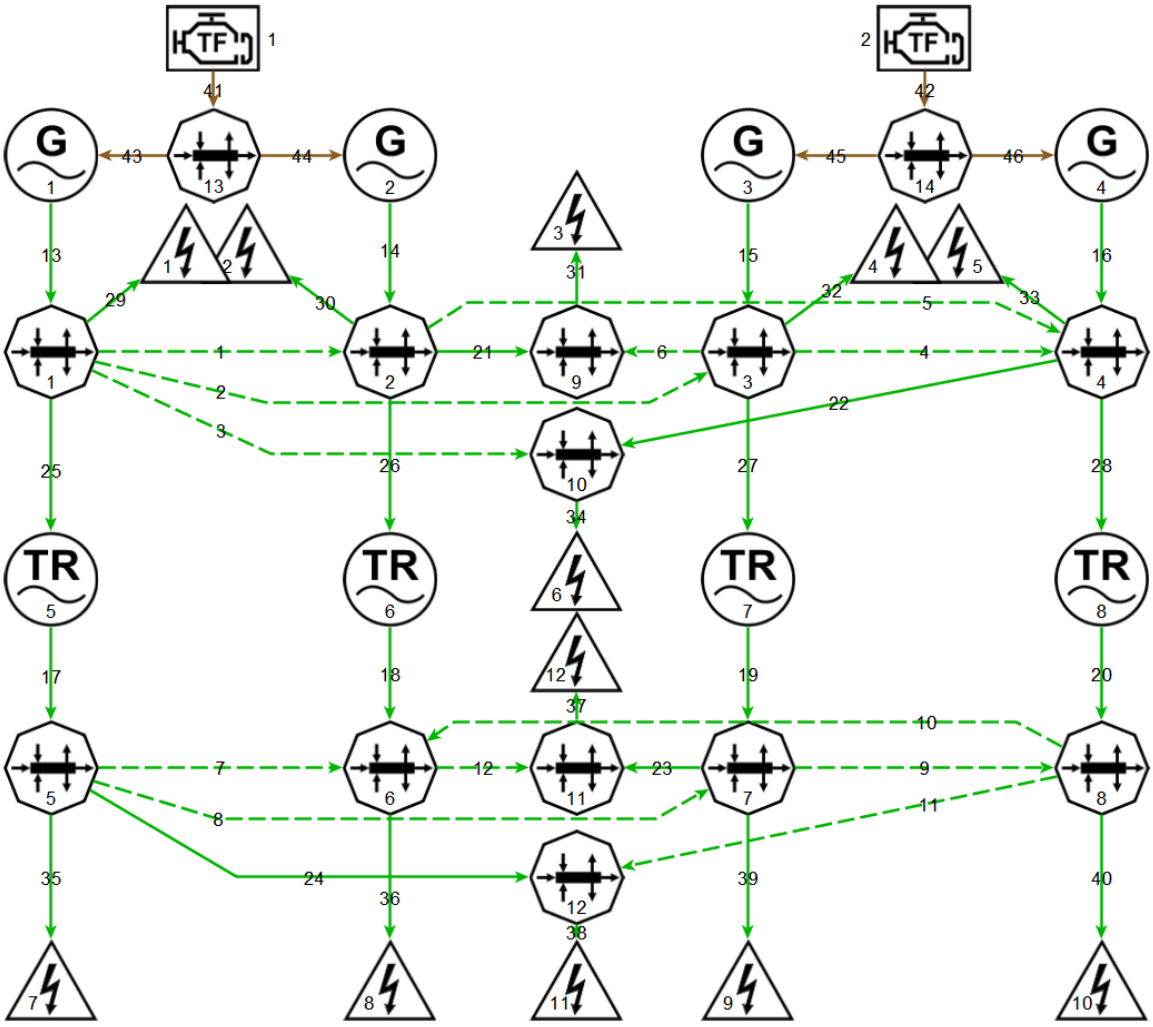


APPENDIX B - LARGE TEST ARCHITECTURE

B.1 - MINIMAL HyPSO NUMBERING



B.2 - FULL HyPSO NUMBERING



APPENDIX C – HONEYWELL HVAC EPS [15]

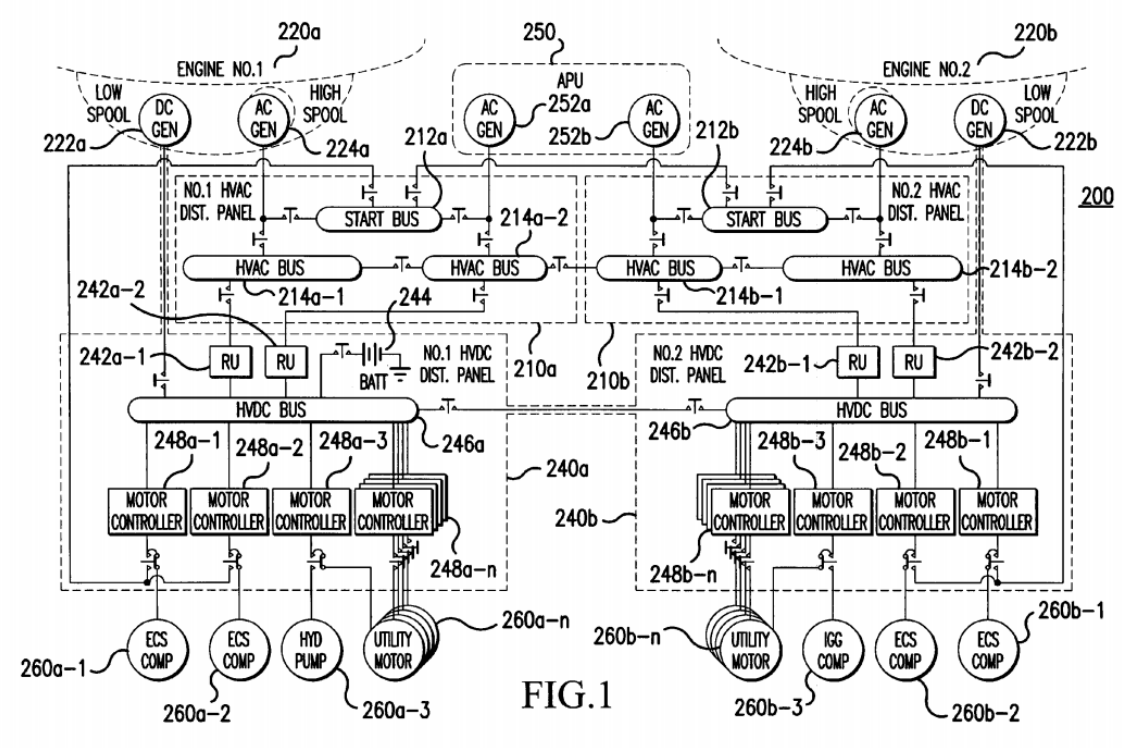


FIG.1

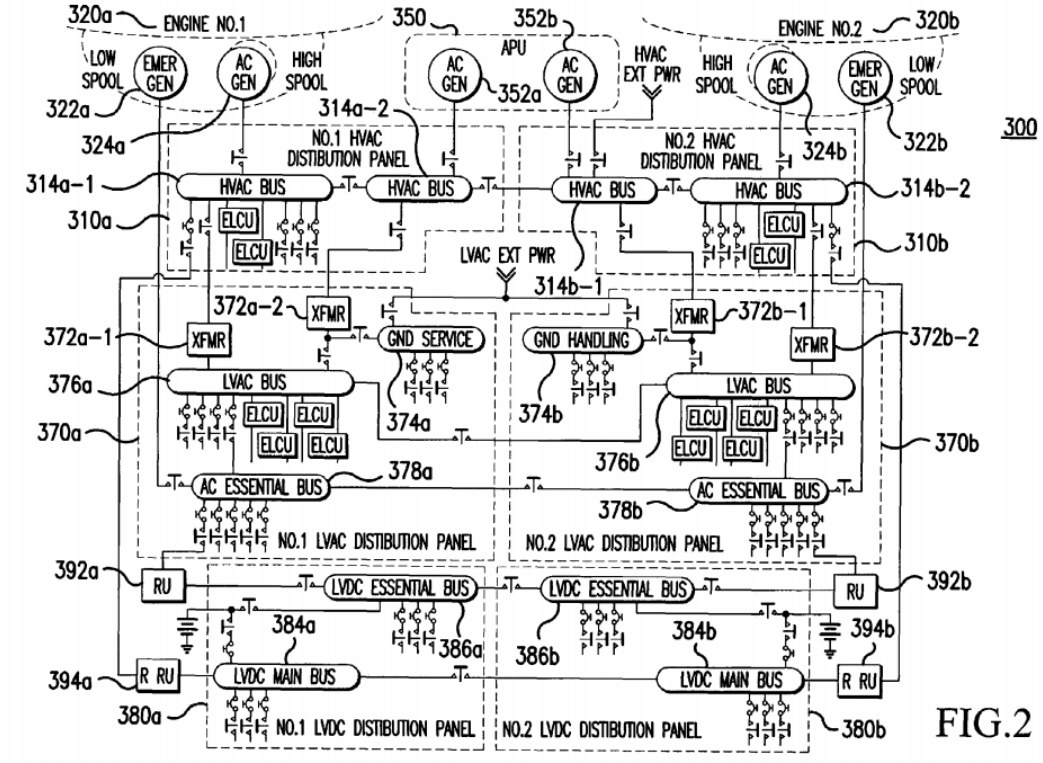
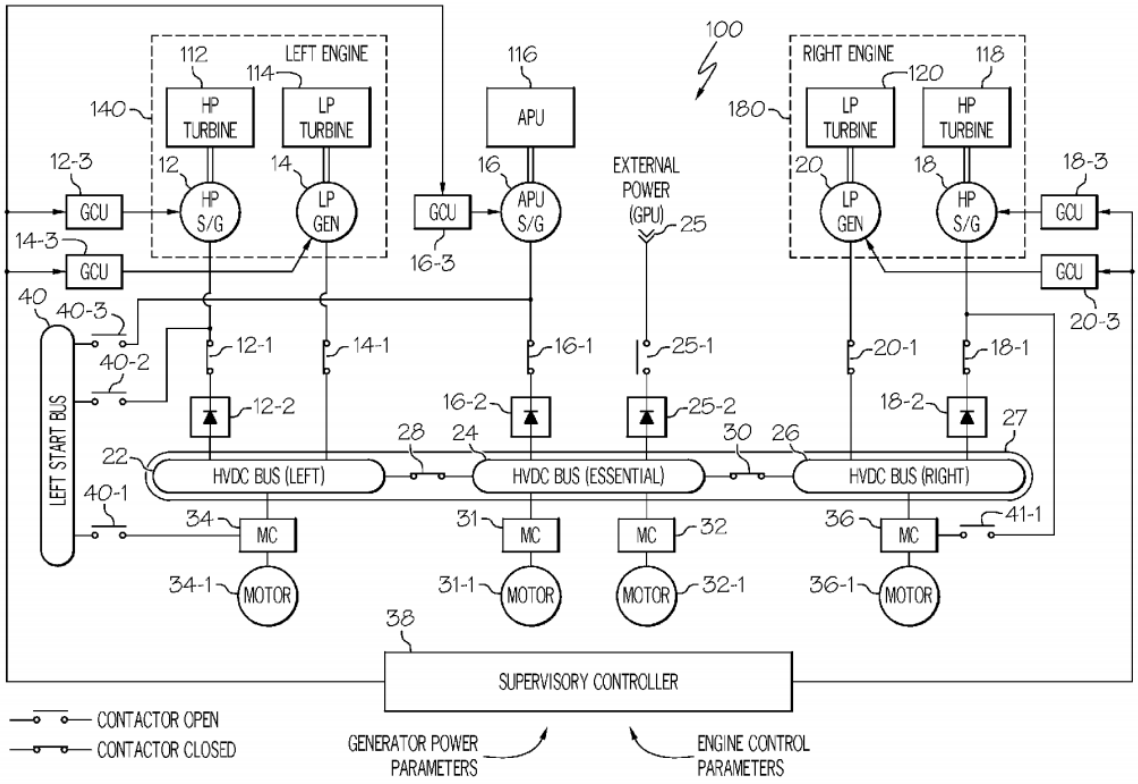


FIG.2

APPENDIX D – HONEYWELL HVDC EPS [16]



APPENDIX E – MATLAB CODE

E.1 – “main.m”

```
% Set up workspace
clear
%load('small_test_arch_workspace_full.mat');
%load('test_arch_workspace.mat')
load('small_test_woeng_workspace.mat');

%close the GUI when it opens after opening the workspace
close force all


% USER INPUT VALUE - integer or row vector
% Number of flight hours over which to generate the state
% probabilities. This value may also be obtained from the Mission tab
% in the HyPSO tool.
%   Example:
% simFlightHours = sum(H.f2tMISStable.Data(:,7))/360;
%    Calculates the state probabilities for the same length as the
%    current HyPSO mission.
% If a Row vector is input, state probabilities will be generated for
% multiple time values.
%   Example:
% simFlightHours = 1:100;
%    Generates a row vector with 100 time steps incrementing by 1.
simFlightHours = 1:10;

% USER INPUT VALUES - logical true/false
% User decided whether to include engines, machines, and buses in the
% failure simulations. If the system is included, it MUST have a
% corresponding FailRate column vector. If includeBuses is true, the
% busTypes and busInputPaths below must be declared for each bus for
% which failures are to be simulated.
includeEngines = false;
includeMachines = true;
includeBuses = true;

% USER INPUT VALUE - logical true/false
```

```
% When reporting the available power, take the difference from nominal
% available power.
takePwrDiffFromNom = true;

% USER INPUT VALUE - column vectors
% Since buses are abstracted to routing nodes in HyPSO, we can declare
% bus names individually for clarity. Input as column cell array. User
% must declare the bus's main power input path in busInputPath. This
% is the path that will be turned to 0 when the bus is supposed to
% fail.
%
% ASSUMPTIONS
% 1 Routing nodes are numbered in the same sequence as they are
%   declared below!
% 2 busInputPath is a column vector containing path numbers that are
%   in corresponding order with busTypes
busTypes = {'AC'; 'AC'; 'DC'; 'DC'};
busInputPath = [3; 4; 9; 10];
% busTypes = {'AC';'AC';'AC';'AC';'DC';'DC';'DC';'DC';...
%             'ACESS';'ACSHED';'DCESS';'DCSHED'};
% busInputPath = [7; 8; 9; 10; 29; 30; 32; 33; 19; 26; 37; 41];


% USER INPUT VALUE - column vectors
% This variable can be added to the Engine tab in HyPSO
% Each component set to fail needs a specific fail rates
% Input fail rates in the same order as the components are numbers
%engFailRates = [ 1E-7; 1E-7 ];
% machFailRates = [ linspace(7E-4,10E-4,length(simFlightHours));...
%                   linspace(7E-4,10E-4,length(simFlightHours));...
%                   linspace(8E-5,12E-5,length(simFlightHours));...
%                   linspace(8E-5,12E-5,length(simFlightHours))];
% busFailRates = [  linspace(1E-6,1E-6,length(simFlightHours));...
%                   linspace(1E-6,1E-6,length(simFlightHours));...
%                   linspace(1E-6,1E-6,length(simFlightHours));...
%                   linspace(1E-6,1E-6,length(simFlightHours))];
machFailRates = [7E-4;7E-4;8E-5;8E-5];
busFailRates = [1E-6;1E-6;1E-6;1E-6];
% engFailRates = [1E-7;1E-7];
% machFailRates = [8E-4;5E-4;6E-4;7E-4;1E-5;9E-5;3E-5;8E-5];
% busFailRates = [1E-7;7E-7;1E-6;4E-7;1E-6;...
```

```
%                 9E-7;3E-6;4E-7;1E-8;1E-7;2E-8;3E-7];


% Build the sys struct that describes the architecture
buildSys;
% Construct state probability data structure
states = stProb_rev4 ( sys, simFlightHours );
% Add together redundant states
states = aggregate(states);
% Eliminate trivial 2fail states
states = elimTrivial(states,'Generator','AC','Transformer');
% Correct status numbers on each of the states
[states,H.f2tSTATgARCHtable.Data] = statusize(states, busInputPath,...
    numEng, numMach, H.f2tSTATgARCHtable.Data);
% Attach available power from HyPSO simulation to the states struct
states = getPwrAvail(states, route, path, takePwrDiffFromNom);

% Clean up interface and workspace
clear busInputPath sys simFlightHours
clear numEng numMach takePwrDiffFromNom
```

E.2 – “buildSys.m”

```
%---------------------------------
% This script builds a system struct that contains all of the
% pertinent information about the system architecture that is needed
% to create the state space. This information is taken directly from
% the HyPSO tool GUI - the tables in the H struct. Data describing the
% engines and machines are taken from H.f2tENGtable.Data and
% H.f2tMACHtable.Data respectively.
%
% User must choose to include engines, machines, and buses in the
% failure state space in main.m by setting includeEng, includeMach,
% and include.Bus.
%
% ASSSUMPTIONS
% A HyPSO simulation has already been run with the model under test so
% that the workspace is populated with the H struct.
```

```
%---------------------------------


% Build system architecture struct
sys = struct('sub',[],'subnum',[],'type',[],'num',[],'rate',[]);

% Create vectors for counting the unique subsystem types
sysType = unique( [H.f2tMACHtable.Data(:,9); ...
                   H.f2tENGtable.Data(:,2); busTypes] );
sysCount = ones(size(sysType,1),1);

% Build engine section
numEng = size(H.f2tENGtable.Data,1);
if includeEng
    for x = 1:numEng
        sys(x).sub = 'ENG';
        sys(x).subnum = x;
        sys(x).type = H.f2tENGtable.Data(x,2);
        sys(x).rate = engFailRates(x,:);
        % Keep a running tally of the number of engine types
        %(i.e. 'Turbofan')
        [~,y] = ismember(sys(x).type,sysType);
        sys(x).num = sysCount(y);
        sysCount(y) = sysCount(y) + 1;
    end
end

% Build machine section
numMach = size(H.f2tMACHtable.Data,1);
if includeMach
    for x = 1:numMach
        sys(x+numEng).sub = 'MACH';
        sys(x+numEng).subnum = x;
        sys(x+numEng).type = H.f2tMACHtable.Data(x,9);
        sys(x+numEng).rate = machFailRates(x,:);
        % Keep a running tally of the number of machine types
        % (i.e. 'Generator', 'Transformer')
        [~,y] = ismember(sys(x+numEng).type,sysType);
        sys(x+numEng).num = sysCount(y);
        sysCount(y) = sysCount(y) + 1;
    end
```

```
end

% Build bus section
if includeBus
    for x = 1:size(busTypes,1)
        sys(x+numEng+numMach).sub = 'BUS';
        sys(x+numEng+numMach).subnum = x;
        sys(x+numEng+numMach).type = busTypes(x);
        sys(x+numEng+numMach).rate = busFailRates(x,:);
        % Keep a running tally of the number of bus types
        % (i.e. 'AC', 'DC', 'HVDC', 'ACESS')
        [~,y] = ismember(sys(x+numEng+numMach).type,sysType);
        sys(x+numEng+numMach).num = sysCount(y);
        sysCount(y) = sysCount(y) + 1;
    end
end

%clean up the workspace
clear x y sysType sysCount busTypes
clear engFailRates machFailRates busFailRates
clear includeEng includeMach includeBus
%------------------------------
```

E.3 – “stProb.m”

```
function [ states ] = stProb_rev4( sys, flightHours)
% stProb This function builds and solves the differential equations to
% find the state probability of each of the possible states in the
% system. Using the system description built in buildSys.m,

rates = repmat([sys(:).rate]',1,size(sys,2)+1);
rates((size(sys,2)+1):(size(sys,2)+1): end) = 0;

count = size(sys,2);
states =
struct('type','NOM','P',[],'tsimhrs',flightHours,'status',1,'avail',[],
'f1',[]);
```

```
syms P(t) double
states.P = dsolve(diff(P) == 0 - sum(rates(:,1))*P, P(0) == 1);

states.f1 = deal(sys);
for x = 1:count
    states.f1(x).P = dsolve(diff(P) == sum(rates(x,1) .* states.P) -
sum(rates(:,x+1))*P, P(0) == 0);
    states.f1(x).f2 = deal(sys);
    for y = 1:count
        states.f1(x).f2(y).P = dsolve(diff(P) == sum(rates(y,x+1) .*
states.f1(x).P), P(0) == 0);
    end
end

nEval = length(flightHours);

for x = 0:nEval-1
    states.P(nEval-x) = subs(states.P(1), t, flightHours(nEval-x));
end
states.P = eval(states.P);
for x = 1:count
    for y = 0:nEval-1
        states.f1(x).P(nEval-y) = subs(states.f1(x).P(1), t,
flightHours(nEval-y));
    end
    states.f1(x).P = eval(states.f1(x).P);
    for y = 1:count
        for z = 0:nEval-1
        states.f1(x).f2(y).P(nEval-z) = subs(states.f1(x).f2(y).P(1),
t, flightHours(nEval-z));
        end
        states.f1(x).f2(y).P = eval(states.f1(x).f2(y).P);
    end
end
end
```

E.4 – “checksum.m”

```
function [  ] = checksum( states )
%checksum Checks the sum of all the state probabilities in the system
```

```
% state data structure to ensure they add up to one. Due to precision
% error in double floating point arithmatic, the error may be
% non-zero. Typical errors are less than 1E-14, a negligible amount.
% Enter eps(1) for the smallest incremental value after 1 for a double
% floating point number.
%
% INPUTS
% states The system state probability data structure

% Subtract the nominal state probability
error = ones(1,length(states.P)) - states.P;

% Subtract 1f and 2f state probabilities
for x = 1:size(states.f1,2);
    error = error - states.f1(x).P;
    for y = 1:size(states.f1(x).f2,2)
        error = error - states.f1(x).f2(y).P;
    end
end

% Throw an error message if error = |sum(P) - 1| is greater than a
% negligable amount of 1E-14
for x = 1:length(error)
    assert(abs(error(x))<1E-14,...
    'Sum of state prob error from 1 greater than |1E-14|: P(%d)=%e'...
    , x, error(x))
end

end
```

E.5 – “elimTrivial.m”

```
function [ states ] = elimTrivial( states, inMach, bus, outMach )
%elimTrivial elimiates trivial architecture states from the state
% space. A trivial state is one that the same load availability,
% available power, and configuration as a previous state. This occurs
% when a machine fails that was supplying, or supplied by, an already
% failed bus.
```

```
%
% Example: A state where AC1 bus fails has the same configuration,
% available power, and load availability as:
%
% 1) A state where AC1 bus and GEN1 fails
% 2) A state where AC1 bus and TRU1 fails
%
% INPUTS
% state The architecture state space data structure
% inMach A string containing the name of the input machine
% bus A string containing the name of the intermediary bus
% outMach A string containing the name of the output machine
% Example
% inMach = 'Generator'
% bus = 'AC'
% outMach = 'Transformer'
%
% OUTPUTS
% state The updated architecture state space data structure with
%   eliminated trivial states
%
% ASSUMPTIONS
% The inMach supplying power to the bus, and the outMach supplied by
% the bus have the same num


% Check each single fail state, looking for the inMach or outMach
for x = 1:size(states.f1,2)
    if strcmp(states.f1(x).type, inMach) ||...
       strcmp(states.f1(x).type, outMach)
        % Store the number of the inMach or outMach
        num = states.f1(x).num;
        % Find the struct index of the 1 bus fail state. The bus has
        % the same num as the inMach/outMach
        y = find(strcmp([states.f1(:).type], bus) &...
                 [states.f1(:).num] == num);
        % Find the struct index of the trivial 2fail state
        z = find(strcmp([states.f1(x).f2(:).type], bus) &...
                 [states.f1(x).f2(:).num] == num);
        % Add the 2fail state probability to the 1fail state
        states.f1(y).P = states.f1(y).P + states.f1(x).f2(z).P;
```

```
        % Delete the 2fail state
        states.f1(x).f2(z) = [];
    end
end

checksum(states);
end
```

E.6 – “aggregate.m”

```
function [ states ] = aggregate( states )
%aggregate This function combines obvious identical system states. For
% example - it combines the state where GEN1 fails, then GEN2 fails
% with the state where GEN2 fails, then GEN1 fails. It doesn't matter
% which path the systemt took to reach the state - it will have the
% same system configuration and characteristics. Therefore, we can
% combine the states by adding their probabilities.


count = size(states.f1,2);

% Add the probabilities of the identical 2 fail states together
for x = 1:count
    for y = x+1:count
        states.f1(x).f2(y).P = states.f1(x).f2(y).P +...
                               states.f1(y).f2(x).P;
    end
end

% Delete the entry of the duplicate 2 fail state.
for x = 1:count
    for y = 1:x
        states.f1(x).f2(1) = [];
    end
end

checksum(states);
end
```

E.7 – “getPwrAvail.m”

```
function [ states ] = getPwrAvail( states, route, path, takeDiff )
%getPwrAvail parses the available power from the HyPSO simulation and
%attaches it to the states data structure.
%
%
% INPUTS
% states The architecture state space data structure
% route The route data structure generated by the HyPSO simulation
% path The path data structure generated by the HyPSO simulation
% takeDiff Logical integer choosing whether to take the difference
%   from nominal before attaching the available power
%
% OUTPUTS
% state The updated architecture state space data structure with
%    attached available power


%-------------------------------------------------
% This section identifies when two routing nodes are connected
% together. Only checks routing nodes that have report available power
% enabled (route.checkPwr). This corrects the error where both routing
% nodes will report the same available power when they are connected
% together. In reality, only one node should have that available
% power, and the other node is just drawing from the first

%initialize variables
outPaths = [];
inPaths = [];
for x = 1:route.n
    %Only check routing nodes with report available power enabled
    if route.checkPwr(x)
        %Create an array of all input and output paths
        outPaths = [ outPaths,route.pathOut(1,x) ];
        inPaths = [ inPaths,route.pathIn(1,x) ];
    end
end
```

```
% This loop finds the input and output paths between routing nodes
% with check available power enabled
for x = 1:route.n
    %Only check routing nodes that have report available power enabled
    if route.checkPwr(x)
        % Check if an output or input path corresponds to another node
        % with available power enabled.
        in(x)  = {intersect( cell2mat( route.pathIn(1,x) ),...
                             cell2mat( outPaths ))};
        out(x) = {intersect( cell2mat( route.pathOut(1,x) ),...
                             cell2mat( inPaths ))};
    else
        % If checkPwr not true, leave an empty cell in the array
        in(x)  = {[]};
        out(x) = {[]};
    end
end

% Convert from a cell array to a matrix
in  = ~cellfun( @isempty,in );
out = ~cellfun( @isempty,out );
%-------------------------------------------------



%Create a copy of the available power table
availDiff = route.pwrAvail;

%
for x = 1:size(states.f1,2)
    temp = states.f1(x).status;
    for y = 1:size(availDiff,2)
        if in(y)  && path.status( temp, in(y) ) ==  1 ||...
           out(y) && path.status( temp, out(y)) == -1

            availDiff(temp,y) = 0;
        end
    end
    for z = 1:size(states.f1(x).f2,2)
        temp = states.f1(x).f2(z).status;
```

```
        for y = 1:size(availDiff,2)
            if in(y)  && path.status( temp, in(y) ) ==  1 ||...
               out(y) && path.status( temp, out(y)) == -1

                availDiff(temp,y) = 0;
            end
        end
    end
end

% Delete cells that have Not A Number (NaN) in them
availDiff = availDiff( :, ~isnan( availDiff(1,:) ));
availDiff = availDiff( ~isnan( availDiff(:,1) ), :);

% Take the difference between the nominal available power and each
% state. Each state will now contain the difference from nominal.
% Nominal remains unchanged. User Enabled
if takeDiff
    for col = 1:size(availDiff,2)
        availDiff(2:end, col) = availDiff(2:end, col) -...
                                availDiff(1,col);
    end
end


% Assign nominal available power to states data structure
states.avail = availDiff(1,:);

% Assign available power to all other states in data structure
for x = 1:size(states.f1,2)
    states.f1(x).avail = availDiff( states.f1(x).status, : );
    for y = 1:size( states.f1(x).f2, 2 )
        states.f1(x).f2(y).avail = ...
            availDiff( states.f1(x).f2(y).status, : );
    end
end

end
```

E.8 – “statusize.m”

```
function [states,atable,etable] = statusize(states,paths,numEng,...
                                            numMach,atable,etable)
%statusize This function builds the architecture states on the status tab.
% It assigns each state in the architecture states data structure a
% "status" that denotes the corresponding architecture status in the HyPSO
% tool. It sets the appropriate components to fail, depending on the
% architecture state. In order to fail bus bars, this function sets the
% main power input path to fail. The user must choose which input path is
% the main input path that will fail the bus bar.
%
% Inputs
% states The architecture states data structureW
% paths The main power input paths that will fail in order to cause a bus
%   bar failure. The paths MUST be in the same order as the subnum
%   numbering for the BUS sub.
% numEng The number of engines
% numMach The number of machines
% atable The HyPSO GUI status arch table H.f2tSTATgARCHtable.Data
% etable The HyPSO GUI status event table H.f2tSTATgEVNTtable1.Data


count = size(states.f1,2);
status = 2;
for x = 1:count
    states.f1(x).status = status;
    atable{status,1} = strcat(states.f1(x).sub, num2str(states.f1(x).subnum));
    if strcmp(states.f1(x).sub,'ENG')
        atable{status,states.f1(x).subnum + 1} = 0;
    elseif strcmp(states.f1(x).sub,'MACH')
        atable{status,states.f1(x).subnum + numEng + 1} = 0;
    else
        atable{status,paths(states.f1(x).subnum) + numEng + numMach + 3} = 0;
    end
    status = status + 1;
end

for x = 1:count
    for y = 1:size(states.f1(x).f2,2)
        if strcmp(states.f1(x).sub,'ENG')
            atable{status,states.f1(x).subnum + 1} = 0;
        elseif strcmp(states.f1(x).sub,'MACH')
            atable{status,states.f1(x).subnum + numEng + 1} = 0;
        else
            atable{status,paths(states.f1(x).subnum) + numEng + numMach + 3} = 0;
        end

        states.f1(x).f2(y).status = status;
        atable{status,1} = strcat(states.f1(x).sub, num2str(states.f1(x).subnum),
states.f1(x).f2(y).sub, num2str(states.f1(x).f2(y).subnum));
```

```
        if strcmp(states.f1(x).f2(y).sub,'ENG')
            atable{status,states.f1(x).f2(y).subnum + 1} = 0;
        elseif strcmp(states.f1(x).f2(y).sub,'MACH')
            atable{status,states.f1(x).f2(y).subnum + numEng + 1} = 0;
        else
            atable{status,paths(states.f1(x).f2(y).subnum) + numEng + numMach + 3} = 0;
        end

        status = status + 1;
    end
end
status = status - 1;
[etable(1,1:status)] = deal(num2cell(linspace(0,(status-1)*100,status)));
[etable(2,1:status)] = deal(num2cell(linspace(1,status,status)));

checksum(states);
end
```

E.9 – “nom_prob_vs_time.m”

```
function clear
load('states_plot_test.mat');
y = states.P;
x = states.tsimhrs;
plot(x,y,'r');
hold on
plot(states.tsimhrs,states.f1(1).P,'g')
%barColorMap = hot(8);
%set(h,'FaceColor', barColorMap(1,:));
plot(states.tsimhrs,states.f1(1).P + sum(vertcat(states.f1(1).f2(:).P)),'b');

title('Probability of No Failures Over Time');
ax = gca;
ax.YScale = 'log';

ax.XLim = [0 10000];
ax.YLim = [0 1];

h.BarWidth = 0.2;

ax.XTickMode = 'auto';
%ax.XTick = [1 10 100 1000];
ax.YTickMode = 'auto';
%ax.YTick = 0:0.1:1;
%ax.XTickLabel = num2str(states.tsimhrs);

ax.YGrid = 'on';
```

```
ax.YMinorGrid = 'on';
ax.XGrid = 'on';

ylabel('State Probability');
xlabel('Flight Time (hours)');
```

E.10 – “precentServiced.m”

```
Pserv = states.P;
Punserv = zeros(1,length(states.P));

for x = 1:size(states.f1,2)
    if opt.constrVio(states.f1(x).status) > 0.1
        Punserv = Punserv + states.f1(x).P;
    else
        Pserv = Pserv + states.f1(x).P;
    end
    for y = 1:size(states.f1(x).f2,2)
        if opt.constrVio(states.f1(x).f2(y).status) > 0.1
            Punserv = Punserv + states.f1(x).f2(y).P;
        else
            Pserv = Pserv + states.f1(x).f2(y).P;
        end
    end
end
figure
plot(1:length(Punserv),Punserv);
title('Probability of At Least One Load Unserviced');
ylabel('Probability');
xlabel('Flight Hours');

set(gca,'YGrid','on',...
        'YMinorGrid','on',...
        'XGrid','on');

clear Pserv Punserv x y
```

E.11 – “bar_3d.m”

```
Pf = vertcat(states.f1(:).P);
Pf(:,1:9) = [];
min = min(Pf);
%P2f = zeros(size(states.f1,2));
for x = 1:size(states.f1,2)
    for y = 1:size(states.f1(x).f2,2)
        Pf(y,x+1) =  states.f1(x).f2(y).P(10);
        if Pf(y,x+1) < min
            min = Pf(y,x+1);
        end
    end
end
Pf = Pf';
figure
h1 = bar3(nan(size(Pf)));
set(h1,'FaceColor',[1 0 0]);
hold on
h2 = bar3(nan(size(Pf)));
set(h2,'FaceColor',[1 0.8 0.2]);
hold on
h3 = bar3(nan(size(Pf)));
set(h3,'FaceColor',[1 1 0]);
```

```
hold on
h4 = bar3(nan(size(Pf)));
set(h4,'FaceColor',[0 1 0]);
hold on
h5 = bar3(nan(size(Pf)));
set(h5,'FaceColor',[0 0 1]);
hold on
l = {'P>1E-3 frequent',...
       '1E-3>P>1E-5 reasonably probable',...
       '1E-5>P>1E-7 remote',...
       '1E-7>P>1E-9 extremely remote',...
       'P<1E-9 extremely improbable'};
legend([h1(1) h2(1) h3(1) h4(1) h5(1)],l,'Position',[0.685 0.772 0.258 0.155]);

h = bar3(Pf);



%---------------------
%This segment of code by Matt Fig posted to MATLAB support forums.
%Allows access to individual bars to change facecolor.
%http://www.mathworks.com/matlabcentral/
%    answers/5424-how-to-colorize-individual-bar-in-bar3
%
cm = get(gcf,'colormap');  % Use the current colormap.
cnt = 0;
for jj = 1:length(h)
    xd = get(h(jj),'xdata');
    yd = get(h(jj),'ydata');
    zd = get(h(jj),'zdata');
    delete(h(jj))
    idx = [0;find(all(isnan(xd),2))];
    if jj == 1
        S = zeros(length(h)*(length(idx)-1),1);
        dv = floor(size(cm,1)/length(S));
    end
    for ii = 1:length(idx)-1
        cnt = cnt + 1;
        S(cnt) = surface(xd(idx(ii)+1:idx(ii+1)-1,:),...
                         yd(idx(ii)+1:idx(ii+1)-1,:),...
                         zd(idx(ii)+1:idx(ii+1)-1,:),...
                         'facecolor',cm((cnt-1)*dv+1,:));
    end
end
%---------------------

for x = 1:size(Pf,1)
    for y = 1:size(Pf,2)
        color = [0 0 1];
        if Pf(y,x) > 1E-3
            color = [1 0 0];
        elseif Pf(y,x) > 1E-5
            color = [1 0.8 0.2];
        elseif Pf(y,x) > 1E-7
            color = [1 1 0];
        elseif Pf(y,x) > 1E-9
            color = [0 1 0];
        elseif Pf(y,x) == 0
            color = [1 1 1];
            set(S((x-1)*size(Pf,2)+y),'edgecolor',[1 1 1]);
        end
        set(S((x-1)*size(Pf,2)+y),'facecolor',color);
    end
end

title('Probability of Single and Double Failures');
xlabel('First Failure');
ylabel('Second Failure');
zlabel('Probability of Failure');


%---------------------
%This segment of code from Mathworks Support for correcting ZScale log
%error on bar3 plots posted to MATLAB support forums
%http://www.mathworks.com/matlabcentral/answers/100500-how-can-i-set-
%      the-zscale-of-a-bar3-plot-to-logarithmic-in-matlab
%
b = get(gca,'Children');
for i = 1:length(b)
```

```
        ZData = get(b(i), 'ZData');
        ZData(ZData==0) = min/10;
        set(b(i), 'ZData', ZData);
end
%--------------------

for x = 1:size(Pf,2)
    xl{x} = strcat(states.f1(x).sub,num2str(states.f1(x).subnum));
end
yl = xl;
yl{1} = 'Only one failure';
set(gca,'ZScale','log',...
        'ZLim',[min/10 1],...
        'ZTick',[1E-9 1E-7 1E-5 1E-3 1],...
        'XTick',1:size(Pf,2),...
        'YTick',1:size(Pf,2),...
        'YTickLabel',yl,...
        'XTickLabel',xl,...
        'YTickLabelRotation',35,...
        'XTickLabelRotation',325,...
        'Xdir','reverse',...
        'Ydir','reverse'...
        );
set(gcf,'Position',[200 200 750 600]);
clear h1 h2 h3 h4 h5 i idx ii jj cm cnt color dv b h x y min l
clear xd yd zd ZData xl yl S Pf
```

E.12 – “bar_2d_avail_2fail.m”

```
clear ndx x y ndx2 barColorMap xl st av
count = size(states.f1,2);
totalFails = states.f1(count-1).f2.status - 1;
x = (1:totalFails)';

for ndx = 1:count
    xl{ndx} = strcat(states.f1(ndx).sub, num2str(states.f1(ndx).subnum));
end


for ndx = 1:count
    for ndx2 = 1:size(states.f1(ndx).f2,2)
        xl{end+1} = strcat(states.f1(ndx).sub,...
                           num2str(states.f1(ndx).subnum),...
                           states.f1(ndx).f2(ndx2).sub,...
                           num2str(states.f1(ndx).f2(ndx2).subnum));
    end
end

for ndx = 1:count
    y(ndx) = states.f1(ndx).P(10);
    if opt.constrVio(states.f1(ndx).status) > 0.1
        barColorMap(ndx,:) = [1,0,0];
    else
        barColorMap(ndx,:) = [0,1,0];
%         diff = sum(states.avail) - abs(sum(states.f1(ndx).avail));
%         diff = diff/sum(states.avail);
%         if sum(states.f1(ndx).avail) > 0;
%             barColorMap(ndx,:) = [diff,0,0];
%         else
%             barColorMap(ndx,:) = [0,diff,0];
%         end
    end

end

for ndx = 1:count
    for ndx2 = 1:size(states.f1(ndx).f2,2)
        y(end+1) = states.f1(ndx).f2(ndx2).P(10);
        if opt.constrVio(states.f1(ndx).f2(ndx2).status) > 0.1
            barColorMap(end+1,:) = [1,0,0];
```

```
        else
            barColorMap(end+1,:) = [0,1,0];
        end
    end
end

figure
subplot(2,1,1);
bar(1,nan,'g');
hold on
bar(1,nan,'r');
hold on
legend('All loads available','At least 1 load unavailable');

for ndx = 1 : totalFails
       % Plot one single bar as a separate bar series.
       h(ndx) = bar(x(ndx), y(ndx), 'BarWidth', 0.9);
       % Apply the color to this bar series.
       set(h(ndx),'FaceColor', barColorMap(ndx,:));
       % Place text atop the bar
       %barTopper = sprintf('%.1e', y(ndx));
       %text(x(ndx)-0.4, y(ndx)+0.3*y(ndx), barTopper, 'FontSize', 8);
       hold on;
end
title('Single and Double Failure States');
ylabel('State Probability');
set(gca,'YScale','log',...
        'XTickMode','manual',...
        'XTick',1:totalFails,...
        'XTickLabel','',...
        'YTickMode','auto',...
        'YGrid','on',...
        'YMinorGrid','on',...
        'XGrid','off');




numGen = length(states.avail);

xpos = zeros(totalFails,numGen);
xneg = zeros(totalFails,numGen);

for ndx = 1:count
    for ndx2 = 1:numGen
        if states.f1(ndx).avail(ndx2) > 0
            xpos(ndx,ndx2) = states.f1(ndx).avail(ndx2);
        else
            xneg(ndx,ndx2) = states.f1(ndx).avail(ndx2);
        end
    end
end

for ndx = 1:count
    for ndx2 = 1:size(states.f1(ndx).f2,2)
        st = states.f1(ndx).f2(ndx2).status - 1;
        for ndx3 = 1:numGen
            av = states.f1(ndx).f2(ndx2).avail(ndx3);
            if av > 0
                xpos(st,ndx3) = av;
            else
                xneg(st,ndx3) = av;
            end
        end
    end
end
subplot(2,1,2);
bar(1:totalFails, xpos, 0.5, 'stack');
hold on
bar(1:totalFails, xneg, 0.5, 'stack');

%title('Available Power at Each Generator');
ylabel('Available Power (kW)');
xlabel('Failed System');
set(gca,'XTickLabelMode','manual',...
        'XTickLabel',xl,...
        'XTickMode','manual',...
        'XTickLabelRotation',90,...
```

```
        'XTick',1:totalFails,...
        'Position',[0.13 0.25 0.775 0.25834]);

for x = 1:numGen
    ll{x} = strcat('Gen',num2str(x));
end

legend(ll,'Location','southeast');

set(gcf,'Position',[1 1 500 750]);

clear count ndx ndx2 x y barColorMap diff ll barTopper
clear ndx3 numGen totalFails av st h xl
% T = struct2table(states);
% T1 = struct2table(states.f1);
%
% for i = 1:size(states.f1,1)
%     T2struct(
%
% T2 = struct2table([states.f1.f2]);
```

E.13 – “bar_2d_avail.m”

```
count = size(states.f1,2);

x = (1:count)';

for ndx = 1:count
    y(ndx) = states.f1(ndx).P(10);
    if opt.constrVio(states.f1(ndx).status) > 0.1
        barColorMap(ndx,:) = [1,0,0];
    else
        barColorMap(ndx,:) = [0,1,0];
%         diff = sum(states.avail) - abs(sum(states.f1(ndx).avail));
%         diff = diff/sum(states.avail);
%         if sum(states.f1(ndx).avail) > 0;
%             barColorMap(ndx,:) = [diff,0,0];
%         else
%             barColorMap(ndx,:) = [0,diff,0];
%         end
    end

end


figure
subplot(2,1,1);
bar(1,nan,'g');
hold on
bar(1,nan,'r');
hold on
legend('All loads available','At least 1 load unavailable');

for ndx = 1 : count
        % Plot one single bar as a separate bar series.
        h(ndx) = bar(x(ndx), y(ndx), 'BarWidth', 0.7);
        % Apply the color to this bar series.
        set(h(ndx),'FaceColor', barColorMap(ndx,:));
        % Place text atop the bar
        barTopper = sprintf('%.1e', y(ndx));
        text(x(ndx)-0.4, y(ndx)+0.3*y(ndx), barTopper, 'FontSize', 8);
        hold on;
end
title('Single Failure States');
ylabel('State Probability');
set(gca,'YScale','log',...
        'XTickMode','manual',...
        'XTick',1:count,...
        'XTickLabel','',...
        'YTickMode','auto',...
        'YGrid','on',...
```

```
        'YMinorGrid','on',...
        'XGrid','off');



for ndx = 1:count
    xl(ndx) = strcat(states.f1(ndx).type, num2str(states.f1(ndx).num));
end




xpos = zeros(count,length(states.avail));
xneg = zeros(count,length(states.avail));
for ndx = 1:count
    for ndx2 = 1:length(states.avail)
        if states.f1(ndx).avail(ndx2) > 0
            xpos(ndx,ndx2) = states.f1(ndx).avail(ndx2);
        else
            xneg(ndx,ndx2) = states.f1(ndx).avail(ndx2);
        end
    end
end

subplot(2,1,2);
bar(1:count, xpos, 0.5, 'stack');
hold on
bar(1:count, xneg, 0.5, 'stack');

%title('Available Power at Each Generator');
ylabel('Available Power (kW)');
xlabel('Failed System');
set(gca,'XTickLabelMode','manual',...
        'XTickLabel',xl,...
        'XTickMode','manual',...
        'XTickLabelRotation',90,...
        'XTick',1:count,...
        'Position',[0.13 0.25 0.775 0.25834]);

for x = 1:length(states.avail)
    ll{x} = strcat('Gen',num2str(x));
end

legend(ll,'Location','southeast');

set(gcf,'Position',[1 1 500 750]);

clear count ndx ndx2 x y barColorMap xl xpos xneg diff ll barTopper

% T = struct2table(states);
% T1 = struct2table(states.f1);
%
% for i = 1:size(states.f1,1)
%     T2struct(
%
% T2 = struct2table([states.f1.f2]);
```

E.14 – “bar_1d.m”

```
close all
count = size(states.f1,2);

x = (1:count)';

for ndx = 1:count
    y(ndx) = states.f1(ndx).P(10);

    if opt.constrVio(states.f1(ndx).status) > 1E-14
        barColorMap(ndx,:) = [0,0,1];
    else
```

```
        diff = sum(states.avail) - abs(sum(states.f1(ndx).avail))
        diff = diff/sum(states.avail)
        if sum(states.f1(ndx).avail) > 0;
            barColorMap(ndx,:) = [diff,0,0];
        else
            barColorMap(ndx,:) = [0,diff,0];
        end
    end

end
%h = bar(b,diag([states.f1(:).P]),'stacked')
%set(gca,'YScale','log')
%set(h(1),'facecolor','r');



%barColorMap = hot(count);

for ndx = 1 : count
        % Plot one single bar as a separate bar series.
        h(ndx) = bar(x(ndx), y(ndx), 'BarWidth', 0.7);
        % Apply the color to this bar series.
        set(h(ndx),'FaceColor', barColorMap(ndx,:));
        % Place text atop the bar
        barTopper = sprintf('%.1e', y(ndx));
        text(x(ndx)-0.4, y(ndx)+0.3*y(ndx), barTopper, 'FontSize', 8);

        hold on;
    grid on;
end

title('Single Failure States');
ax = gca;
ax.YScale = 'log';
ylabel('State Probability');
xlabel('Failed System');

ax.XTickMode = 'manual';
ax.XTick = 1:count;
ax.YTickMode = 'auto';
%ax.YTick = 0:0.1:1;


for ndx = 1:count
    xl(ndx) = strcat(states.f1(ndx).type, num2str(states.f1(ndx).num));
end
ax.XTickLabelMode = 'manual';
ax.XTickLabel = xl;

ax.XTickLabelRotation = 90;

ax.YGrid = 'on';
ax.YMinorGrid = 'on';
ax.XGrid = 'off';


% T = struct2table(states);
% T1 = struct2table(states.f1);
%
% for i = 1:size(states.f1,1)
%     T2struct(
%
% T2 = struct2table([states.f1.f2]);
```

APPENDIX F – TRU DATASHEET [17]

PS462

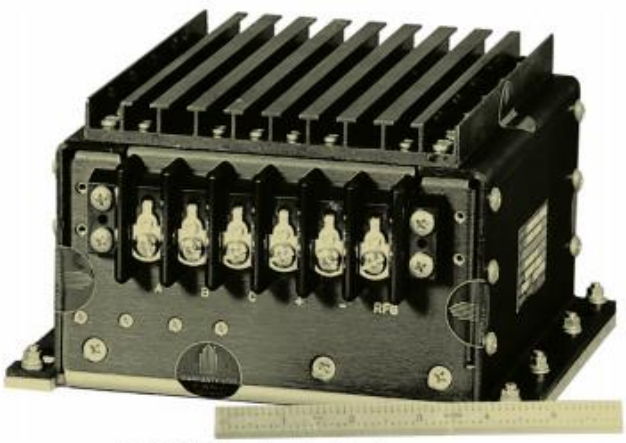
200 Amp Transformer / Rectifier Unit

The Power Paragon PS462 Transformer/ Rectifier Unit (TRU) converts aircraft primary AC power to 28 VDC power, from a three-phase, 115 volt, 400 Hz generator per MIL-STD-704. Designed for minimum space and weight, and maximum reliability, it offers excellent versatility for use in new designs or retrofits.

The TRU design is based on a very simple, time proven concept, using extremely rugged components, which produce both high reliability and extreme overload capabilities of 1200% (2400 amps).

Power Paragon TRUs provide inherent power factor correction on the input lines and low input current harmonic content. These features reduce the demands on the aircraft power generation systems. The output power meets the requirements for 28 volt systems per MIL-STD-704.

This unit is designed to meet the requirements of MIL-C-7115/2B.



200 A, 28 VDC transformer/rectifier unit.

STANDARD FEATURES:

- Selected units are QPL devices
- Inherently rugged design
- No maintenance required
- High output surge capability
- Excellent power factor
- Low harmonic content on input current waveform
- EMI filters on all input and output lines
- Output power conforms to MIL-STD-704 requirements

APPLICATIONS:

- Emergency 28 volt power
- 28 volt aircraft bus

Static Inverters

BASIC SPECIFICATIONS

ELECTRICAL CHARACTERISTICS:

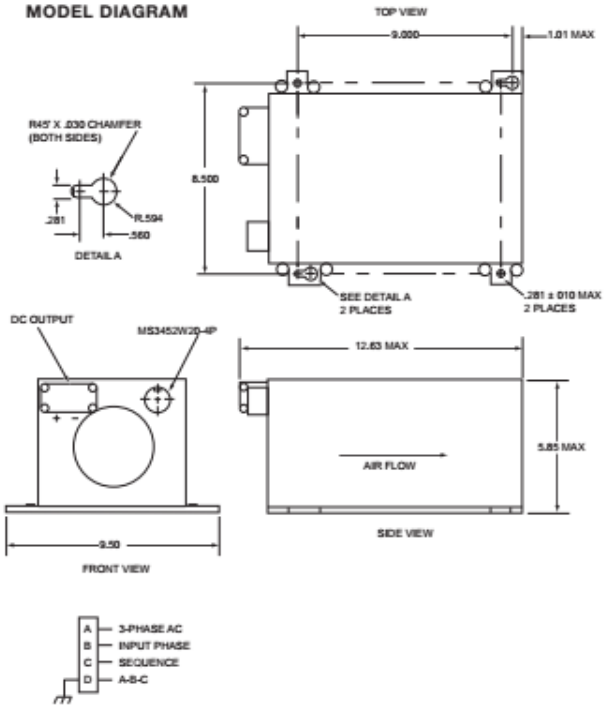
Input

Power	115/200 V, 400 Hz, 3-ph
Voltage	112.5 Vrms to 117.5 Vrms
Frequency	400 Hz, ±20 Hz

Output

Voltage	28 VDC nominal, 25 VDC to32 VDC over full input voltage range and 2A to full load
Current	200 A continuous
Current Overload	500% rated current, 1 sec. 250% rated current, 1 min. 200% rated current, 5 min. followed by 50% load for 10 min.
Ripple	0.53 V peak to peak
Efficiency	80% min. from 25% to . 50% load 85% min. from 51% load to full load
EMI	Conforms to MIL-STD-461, CE03 (150 kHz to 50 MHz with relaxation of 10 dB from 150 kHz to 2 MHz), RE02 (150 kHz to 1 GHz), CS02 (50 kHz to 400 MHz), RS03 (14 kHz to 400 MHz)
Temperature/Altitude	Sea level to 50,000 ft., -55°C per MS33543, Curve II
Vibration	Per MIL-STD-810, Proc. 1A, Method 514.2, except broadband test is not required
Weight	17.5 lbs. max.

MODEL DIAGRAM



Power Paragon
901 E. Ball Road
Anaheim, CA 92805
Tel: 714.956.9200
Fax: 714.956.5397
Email: info.ppi@L-3com.com
L-3Com.com/PowerParagon

Power Paragon

APPENDIX G – CAFTA FTA, SMALL TEST ARCHITECTURE