

AFRL-RI-RS-TR-2008-32
Final Technical Report
February 2008



FAULT TOLERANT AIRBORNE SENSOR NETWORKS FOR AIR OPERATIONS

The Research Foundation of State University of New York at Binghamton

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO COPY

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (<http://www.dtic.mil>).

AFRL-RI-RS-TR-2008-32 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

/s/

ROBERT WRIGHT
Work Unit Manager

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) FEB 2008		2. REPORT TYPE Final		3. DATES COVERED (From - To) May 07 – Oct 07	
4. TITLE AND SUBTITLE FAULT TOLERANT AIRBORNE SENSOR NETWORKS FOR AIR OPERATIONS				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER FA8750-07-1-0172	
				5c. PROGRAM ELEMENT NUMBER 61102F	
6. AUTHOR(S) Eva Wu				5d. PROJECT NUMBER 230S	
				5e. TASK NUMBER 07	
				5f. WORK UNIT NUMBER 01	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The Research Foundation of State University of New York at Binghamton 85 Murray Hill Binghamton NY 13902-6000				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/RISB 525 Brooks Rd Rome NY 13441-4505				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-RI-RS-TR-2008-32	
12. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# WPAFB 08-0279					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT This report summarizes the main results of research conducted during the summer of 2007 on tasking a finite number of cooperative agents to randomly emerging targets for their removal. Faults occur when some agents engaged in a mission are expired. Agents are subject to threat at a level determined by the number of targets present. On the other hand, the rate at which a target is removed depends on the number of cooperative agents assigned to it. Faults effectively change the network architecture and, therefore, degrade the network performance. Designs of control policies that determine the number of agents assigned are based on the network life when expired agents cannot be replenished, and on the network availability when expired agents are replenished at a certain rate. Tasking process is described by a discrete event system in the form of a queuing network, where agents are servers and targets are customers. Optimal policies are determined by solving a Markov decision problem.					
15. SUBJECT TERMS Airborne sensors, multi-agent system, Markov decision problem, fault tolerant networks					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UL	18. NUMBER OF PAGES 74	19a. NAME OF RESPONSIBLE PERSON Robert Wright
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) N/A

Contents

1	Introduction	1
2	Modeling of tasking process	4
3	Design of supervisory control policy	8
4	Performance analysis	11
5	Conclusions	16
6	Designs and simulation modeling	17
6.1	Basic algorithm of the simulation models	17
6.2	Techniques for steady state analysis	18
6.3	Performance analysis from simulation	19
6.3.1	Closed - queue reducible model	19
6.3.2	Closed - queue irreducible model	21
6.3.3	Closed - queue irreducible model with intermittent channel fading	23
6.4	Simulation conclusion	23
7	References	25
8	MATLAB code for the analytic model of the airborne network	27
8.1	Transition rate matrix	27
8.2	Mean time to network expiration	27
8.3	Steady-state availability	29
8.4	Expected response time	31
8.5	Linear Programming	34
9	MATLAB codes and simulations of the airborne Network	38
9.1	Closed queuing network (reducible)	38
9.1.1	Stateflow diagram of the system state (reducible)	39
9.1.2	Stopping rule	40
9.1.3	Computing total number of targets in the system	40

9.1.4	Feedback subsystem	40
9.1.5	Identify system's next state	41
9.1.6	Service rate generator	43
9.1.7	New allocation for return targets	47
9.1.8	Sensor-pair subsystem	49
9.1.9	Failure generation subsystem	50
9.1.9.1	Discrete event subsystem1	51
9.1.9.2	Discrete event subsystem 2	52
9.1.9.3	Discrete event subsystem 3	53
9.2	Irreducible case with system replenishment and with or without channel fading . .	54
9.2.1	Stateflow diagram of the system (irreducible)	55
9.2.2	System up/down stateflow diagram	56
9.2.3	Stopping rule (irreducible)	56
9.2.4	Repairing rule	56
9.2.5	Sensor-pair subsystem (irreducible) without channel fading	57
9.2.6	Failure generation subsystem (irreducible)	58
9.2.6.1	Discrete event subsystem 2	59
9.2.6.2	Discrete event subsystem 3	60
9.3	Irreducible case with system replenishment and intermittent channel fading	61
9.4	Code for running the simulations repeatedly	62
9.4.1	Initial seed value generator:	62
9.4.2	Run simulation repeated to gather results and varying a parameter	63
9.4.2.1	1. Mean time to network expiration	63
9.4.2.2	Response time and availability	64

10 Symbols, Abbreviations and Acronyms

66

List of Figures

1.1	Transmitter location using a TDOA measurement and an FDOA measurement by two airborne sensors.	2
2.1	A queuing network model of a three sensor-pair, finite target population airborne sensor network.	5
2.2	Transitions and transition rates of the tasking process	7
3.1	Control policy indicators. red: one sever/target; green: two servers/target; blue: three servers/target	10
4.1	Mean time to network expiration as a function of target arrival rate with two sets of sensor loss rates as parameters.	12
4.2	Network availability with at least one surviving server.	13
4.3	Comparison between availability with at least one surviving server (dash) and availability with all targets being attended (solid).	15
6.1	Mean time to network expiration vs. arrival rate (simulation model 1)	20
6.2	Expected response time vs. arrival rate (simulation model 2)	21
6.3	Availability vs. arrival rate (simulation model 2)	22
6.4	Expected response time vs. arrival rate (simulation model 3)	23
6.5	Availability vs. arrival rate (simulation model 3)	24
8.1	Transition rate diagram	37
9.1	Top level Simulink block diagram of simulation model 1	38
9.2	System Stateflow diagram of simulation model 1	39
9.3	Stopping rule of simulation model 1	40
9.4	Computing total number of targets	40
9.5	Lower portion subsystem	40
9.6	Identify next state	41
9.7	Service rate generator	43
9.8	New allocation for return targets	47
9.9	Sensor-pair subsystem for simulation model 1	49
9.10	Failure generation subsystem	50
9.11	Discrete event subsystem1	51

9.12	Discrete event subsystem 2	52
9.13	Discrete event subsystem 3	53
9.14	Top level Simulink block diagram for simulation model 2 and 3.	54
9.15	System state-flow diagram for simulation model 2 and 3.	55
9.16	State-flow diagram of network status.	56
9.17	Stopping rule for simulation model 2 and 3.	56
9.18	Repairing rule of simulation model 2 and 3.	56
9.19	Sensor-pair subsystem without channel fading	57
9.20	Failure generation subsystem for simulation model 2 and 3.	58
9.21	This block is within Fig. C.16, the function of this block generates failure time. . .	59
9.22	Discrete event subsystem 3	60
9.23	Sensor-pair subsystem of simulation model 3	61
9.24	Outage and recovery subsystem	62
9.25	Discrete-event subsystem for recovery rate	62

Abstract

This report summarizes the main results of research conducted during the summer of 2007 on tasking a finite number of cooperative agents to randomly emerging targets for their removal. Faults occur when some agents engaged in a mission are expired. Agents are subject to threat at a level determined by the number of targets present. On the other hand, the rate at which a target is removed depends on the number of cooperative agents assigned to it. Faults effectively change the network architecture and therefore degrade the network performance. Designs of control policies that determine the number of agents assigned are based on the network life when expired agents cannot be replenished, and on the network availability when expired agents are replenished at a certain rate. Tasking process is described by a discrete event system in the form of a queuing network, where agents are servers and targets are customers. Optimal policies are determined by solving a Markov decision problem. To facilitate the readers understanding of the motivation, and of the problem, the agents are specialized to networked pairs of airborne sensors that are tasked to locate non-cooperating microwave transmitters as targets.

This work has resulted in a Master's thesis by Yan Guo whose graduate study was partially supported by this grant, a paper submitted to 2008 IFAC World Congress, and another paper submitted to *International Journal of Control, Automation, and Systems*.

Some simulation results are also included. These results are not fully satisfactory and are incomplete. Continued work is desired.

1 Introduction

This work considers tasking a finite number of cooperative agents to randomly emerging targets for their removal. It focuses on enhancing the network performance in the face of expiration of its agents. The resulting network is said to be fault-tolerant. When the agents considered are networked airborne sensors, the mobility and a multiplicity of the sensing nodes make fault-tolerance possible. Fault-tolerant tasking in this work is achieved by implementing operation policies optimized for network availability.

Control of networked multiple agents has been an intensively discussed topic recently in the controls literature [1]. [12], for example, describes a pursuit evasion game, where mobile agents are to chase and capture multiple moving targets in a minimum amount of time, and a network of stationary sensors serves to help enhance the target observability in the game.

With unmanned aerial vehicles (UAVs) replacing stationary networks and manned vehicles, significant improvements in network performance can be expected. Networking in a hostile environment, however, poses new challenges. Data exchange inherent to a networked operation and prolonged mission time due to poor execution expose the otherwise passive location sensors, thus increase the likelihood of the vehicles being destroyed.

Examine a situation where the motion of two unmanned aerial vehicles (UAV) and a hostile radar lie within a plane, as illustrated in Fig. 1.1. Let us assume that the two vehicles are equipped to acquire both the time difference of arrival and the frequency difference of arrival of the radar signal [6]. The sensors are mounted on the vehicles, and thus are subject to the same speed and curvature constraints as that of the vehicles. The sensors are passive nodes when acquiring data from the transmitter, but become active when exchanging data between them in order to provide a location estimate.

It can be seen from Figure 1 that at least two sensor carrying vehicles, which make both a TDOA measurement and an FDOA measurement, are needed in a 2-dimensional setting to locate the target. A noiseless measurement of time difference of arrival by a pair of sensors on the two vehicles is given by

$$\begin{aligned} s_T &= \frac{1}{c} [\sqrt{(x_2 - x_e)^2 + (y_2 - y_e)^2} \\ &\quad - \sqrt{(x_1 - x_e)^2 + (y_1 - y_e)^2}], \end{aligned} \tag{1.1}$$

and a noiseless measurement of frequency difference of arrival by the same pair of sensors is given

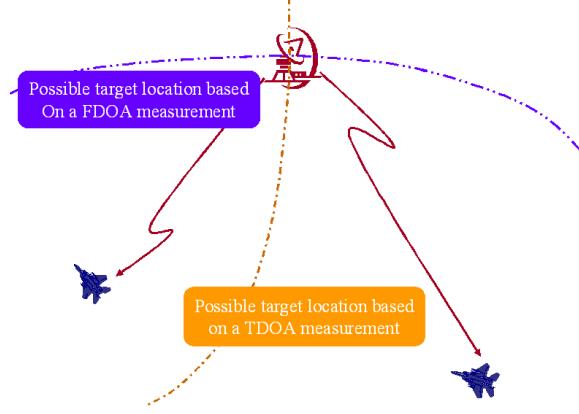


Figure 1.1: Transmitter location using a TDOA measurement and an FDOA measurement by two airborne sensors.

by

$$s_F = \frac{f_e}{c} \left[\frac{(x_2 - x_e)u_2 + (y_2 - y_e)v_2}{\sqrt{(x_2 - x_e)^2 + (y_2 - y_e)^2}} - \frac{(x_1 - x_e)u_1 + (y_1 - y_e)v_1}{\sqrt{(x_1 - x_e)^2 + (y_1 - y_e)^2}} \right], \quad (1.2)$$

where (x_e, y_e) is the transmitter location to be estimated, (x_1, y_1) and (x_2, y_2) are the positions of the two vehicles, respectively, (u_1, v_1) and (u_2, v_2) are the velocities of the vehicles, f_e is the carrier frequency of the transmitted signal, and c is the speed of light.

Since the measurements are always noisy, multiple measurements are needed for an accurate location estimation of the emitter. Such measurements can be distributed temporally along the trajectories of motion of a pair of sensors, or spatially over multiple pairs of sensors, or both. Measurements made by multiple pairs of sensors, which form a network, offer greater degree of fault-tolerance, and greater potential for improved speed and accuracy in target location. The reader is referred to [6] and [14] for more detailed discussion on methods for location estimation and accuracy analysis.

A tasking problem that is specific to this application refers to that of allocating a finite number of sensor pairs to randomly emerging microwave transmitters to maximize the network availability. A tasking policy that is too greedy tends to exhaust resources before the arrival of unanticipated radars, whereas a tasking policy that is too conservative tends to lengthen the exposure of the sensor carrying vehicles. Tasking is treated as a server allocation problem of a queuing network. Optimal policies are sought as the solutions of Markov decision problems.

The report is organized as follows. Chapter 2 modeling the tasking process for a small scale sensor network. Chapter 3 designs supervisory control policies for optimal tasking for the cases where lost sensors can and cannot be replenished by solving appropriate Markov decision problems. Chapter 4 evaluates the network performance in terms of expected network life and steady-

state availability. Chapter 5 concludes the report.

2 Modeling of tasking process

An optimized tasking is one that maximizes the expected life of the network where the lost airborne sensors cannot be replenished, or one that maximizes the expected steady-state availability of the network where the lost sensors can be replenished. In this study fault-tolerance refers to the network's tolerance to vehicle loss.

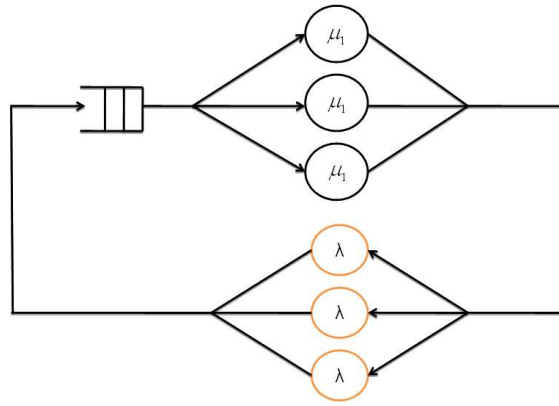
Figure 2.1 is a queuing network model of a six-sensor, finite target population tasking process, where each server represents a pair of sensors capable of independently locating a target to a certain accuracy in the absence other pairs, and a customer is a randomly emerging target.

Each customer resides in the queue or a server is regarded as a detected target which is being or to be served by one or more servers or sensor-pairs. Service is complete as soon as the target location is determined to a required accuracy. A target is then considered removed. A sensor-pair allocated to a target is tied to the target until its service is complete, or the life of the sensor-pair is terminated, whichever comes first. The three delay elements of average delay $1/\lambda$ imply that target population is limited by three at any given time. A new target is generated or replenished at a delay element with rate λ upon the service completion of a target at one or multiple servers.

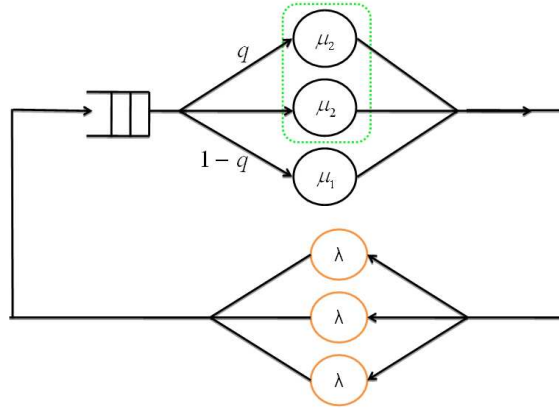
An supervisory control policy determines whether to allocate one, or two, or three pairs of sensors to each reported target, with a corresponding mean service time of $1/\mu_1, (\geq)1/\mu_2, (\geq)1/\mu_3$, respectively, where μ_i denotes the service rate of committing i pairs of sensors to a target. Given the sensing mechanism, the mean service time by a single pair of sensors is in the range of seconds to tens of seconds, dominated by the time required to adjust sensor positions and velocities for continued data collection, exchange, and processing needed for target location to a required accuracy. Each sensor-pair has a mean lifetime $1/\nu_0 \geq 1/\nu_1 \geq 1/\nu_2 \geq 1/\nu_3$, depending on the threat level quantified by the number of targets present as indexed by the subscript. $1/\nu_0$ is the server life representing the expected natural endurance of a vehicle, which is "often an hour or so at best" [13]. It also reflects sensor lives affected by undetected targets. The network is said to have expired when there is no longer a single surviving sensor-pair.

Tasking process model is built in this study with the premise that event life distributions have been established for the process of target arrival ($\exp(\lambda) \equiv 1 - e^{-\lambda t}$), the process of target location ($\exp(\mu_i)$), the process of loss of a sensor-pair ($\exp(\nu_i)$), and the process of sensor replenishment ($\exp(\omega)$) when new sensor carrying vehicles are supplied for an expired network. Since all event lives are assumed to be exponentially distributed, the database unit can be conveniently modeled as a Markov chain specified by a state space $\mathcal{X}$, an initial state probability mass function (pmf) $\pi_x(0)$, and a set of state transition rates λ, μ_i, ν_i , and ω .

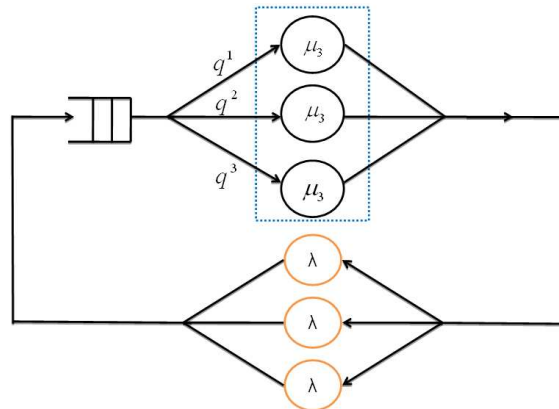
A state name is coded with a 4-digit number indicative of the number of targets present and



(a) One sensor-pair/target allocation



(b) Two sensor-pair/target allocation



(c) Three sensor-pair/target allocation

Figure 2.1: A queueing network model of a three sensor-pair, finite target population airborne sensor network.

the network configuration. A valid state representation is given by QS , where queue length $Q \in \{0, 1, 2, 3\}$, and server state $S = (i, j, k)$, with $i \in \{0, 1, 2, 3, 4\}$, and $j \in \{0, 1, 2, 3, 4, 5\}$, and $k \in \{0, 1, 3, 4, 5\}$. A server state “0”, represented by the value of i , or j , or k , indicates an idle sensor-pair, a “1” indicates a target’s being located by one server (or one sensor-pair), a “2” and a “3” indicate that a target’s being located by two and three cooperating servers, respectively, a “4” indicates a lost server, and a “5” indicates that the lost server has been tied to another server in serving a target. The expired network requires 4 distinct states to memorize the possible queue length distributions. Note that this state specification has assumed homogeneous sensor-pairs and homogeneous targets, and has made use of the symmetry which results in 37 states. A set of alternative state names are assigned from $\mathcal{X} = \{1, 2, \dots, 37\}$ with 0000 mapped to $x = 1$ and the network expiration states mapped to $x = 34, 35, 36$, and 37.

Events that trigger the transitions and the corresponding transition rates are given as follows. An emerging target enters with rate $(3 - Q) \times \lambda$. A target is located by one sensor-pair with rate μ_1 , and $i(> 1)$ cooperative sensor-pairs with rate μ_i . In the latter case, the i servers are configured as a single hyper-exponential server with i parallel stages [4]. An arriving target enters any one of the servers with probability $1/i$, which has a service time distribution $\exp(\mu_i)$. When service is completed, the target is removed, while no new target can enter service when the hyper-exponential server is busy. The service time distribution of a hyper-exponential server is

$$F_i(t) = \sum_{j=1}^i \frac{1}{i} (1 - e^{-\mu_j t}) = 1 - e^{-\mu_i t}, \quad (2.1)$$

which assumes homogeneity of the servers. Loss of a sensor-pair occurs at rate $m\nu_0$ when the network is idle with m remaining sensor-pairs, $m\nu_1$ when one target emerges, and $m\nu_i$ when $i(> 1)$ targets emerge. Replenishment process begins at the network expiration with rate ω . If one of the sensor-pairs is lost while locating a target with other sensor-pairs, the surviving sensor-pairs continue to locate the target at the same rate. This is a simple way to memorize the service already being provided without resorting to a more complex model.

Let $X \in \mathcal{X}$ denote the random state variable at time t . The set of state transition functions is given by

$$p_{i,j}(t) \equiv P[X(t) = j | X(0) = i], \quad i, j = 1, 2, \dots, 37. \quad (2.2)$$

The continuous-time Markov chain can be solved from the forward Chapman-Kolmogorov equation [4],[9]

$$\dot{P}(t) = P(t)Q(u(x)), \quad P(0) = I, \quad P(t) = [p_{i,j}(t)] \quad (2.3)$$

and $Q(u(x))$ is called an infinitesimal generator or a rate transition matrix whose $(i, j)^{th}$ entry is given by the rate associated with the transition from current state i to next state j . Table 1 summarizes information contained in transition rate matrix $Q(u(x))$. Control variable $u(x)$ will be discussed shortly. State probability mass function at time t

$$\pi(t) = [\pi_1(t) \ \pi_2(t) \ \cdots \ \pi_{37}(t)], \quad t \geq 0 \quad (2.4)$$

can be solved from

$$\dot{\pi}(t) = \pi(t)Q(u(x)), \quad \text{given } \pi(t = 0). \quad (2.5)$$

A Markov chain for the tasking process of Figure 2 has been established so far. Since transition rate matrix Q is dependent on control actions, the state transition functions $p_{i,j}(t)$ are being controlled, and so are the state probabilities. Rate transition matrix Q is given in the form of a table in Table 1.

$\mathcal{X}$	Coded $\mathcal{X}$				Arrivals						Completions						Losses & Replenishments					
	Q	S1	S2	S3	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate	$\mathcal{X}'$	Rate
1	0	0	0	0	4	$3^*\lambda^*u1$	7	$3^*\lambda^*u2$	9	$3^*\lambda^*u3$	x	x	x	x	x	x	2	3^*v_0	x	x	x	x
2	0	4	0	0	5	$3^*\lambda^*u1$	8	$3^*\lambda^*u2$	x	x	x	x	x	x	x	x	3	2^*v_0	x	x	x	x
3	0	4	4	0	6	$3^*\lambda$	x	x	x	x	x	x	x	x	x	x	34	v_0	x	x	x	x
4	1	1	0	0	10	$2^*\lambda^*u1$	13	$2^*\lambda^*u2$	x	x	0	μ_1	x	x	x	x	5	3^*v_1	x	x	x	x
5	1	1	4	0	11	$2^*\lambda$	x	x	x	x	2	μ_1	x	x	x	x	6	2^*v_1	x	x	x	x
6	1	1	4	4	12	$2^*\lambda$	x	x	x	x	3	μ_1	x	x	x	x	35	v_1	x	x	x	x
7	1	2	2	0	13	$2^*\lambda$	x	x	x	x	0	μ_2	x	x	x	x	22	$2^*v_1^*u1$	8	v_1^*u2	x	x
8	1	2	2	4	14	$2^*\lambda$	x	x	x	x	2	μ_2	x	x	x	x	23	2^*v_1	x	x	x	x
9	1	3	3	3	15	$2^*\lambda$	x	x	x	x	0	μ_3	x	x	x	x	26	3^*v_1	x	x	x	x
10	2	1	1	0	16	λ	x	x	x	x	4	μ_1	x	x	x	x	11	3^*v_2	x	x	x	x
11	2	1	1	4	17	λ	x	x	x	x	5	μ_1	x	x	x	x	12	2^*v_2	x	x	x	x
12	2	1	4	4	18	λ	x	x	x	x	6	μ_1	x	x	x	x	36	v_2	x	x	x	x
13	2	2	2	1	19	λ	x	x	x	x	7	μ_1^*u1	4	μ_2^*u2	x	x	24	$2^*v_2^*u1$	14	v_2^*u2	x	x
14	2	2	2	4	20	λ	x	x	x	x	5	μ_2^*u1	8	μ_2^*u2	x	x	30	2^*v_2	x	x	x	x
15	2	3	3	3	21	λ	x	x	x	x	4	μ_3^*u1	7	μ_3^*u2	9	μ_3^*u3	28	3^*v_2	x	x	x	x
16	3	1	1	1	x	x	x	x	x	x	10	μ_1	x	x	x	x	17	3^*v_3	x	x	x	x
17	3	1	1	4	x	x	x	x	x	x	11	μ_1	x	x	x	x	18	2^*v_3	x	x	x	x
18	3	1	4	4	x	x	x	x	x	x	12	μ_1	x	x	x	x	37	v_3	x	x	x	x
19	3	2	2	1	x	x	x	x	x	x	10	μ_2^*u1	13	μ_2^*u2	x	x	25	$2^*v_3^*u1$	20	v_3^*u2	x	x
20	3	2	2	4	x	x	x	x	x	x	11	μ_2^*u1	14	μ_2^*u2	x	x	33	2^*v_3	x	x	x	x
21	3	3	3	3	x	x	x	x	x	x	10	μ_3^*u1	13	μ_3^*u2	15	μ_3^*u3	31	3^*v_3	x	x	x	x
22	1	2	5	0	24	$2^*\lambda$	x	x	x	x	2	μ_2	x	x	x	x	6	v_1^*u1	23	v_1^*u2	x	x
23	1	2	5	4	30	$2^*\lambda$	x	x	x	x	3	μ_2	x	x	x	x	35	v_1	x	x	x	x
24	2	2	5	1	25	λ	x	x	x	x	22	μ_1^*u1	5	μ_2^*u2	x	x	12	v_2^*u1	30	v_2^*u2	x	x
25	3	2	5	1	x	x	x	x	x	x	24	μ_1^*u1	11	μ_2^*u2	x	x	18	v_3^*u1	33	v_3^*u2	x	x
26	1	3	3	5	28	$2^*\lambda$	x	x	x	x	2	μ_3	x	x	x	x	27	2^*v_1	x	x	x	x
27	1	3	5	5	29	$2^*\lambda$	x	x	x	x	3	μ_3	x	x	x	x	35	v_1	x	x	x	x
28	2	3	3	5	31	λ	x	x	x	x	5	μ_3^*u1	8	μ_3^*u2	x	x	29	2^*v_2	x	x	x	x
29	2	3	5	5	32	λ	x	x	x	x	6	μ_3	x	x	x	x	36	v_2	x	x	x	x
30	2	2	5	4	33	λ	x	x	x	x	6	μ_2	x	x	x	x	36	v_2	x	x	x	x
31	3	3	3	5	x	x	x	x	x	x	11	μ_3^*u1	14	μ_3^*u2	x	x	32	2^*v_3	x	x	x	x
32	3	3	5	5	x	x	x	x	x	x	12	μ_3	x	x	x	x	37	v_3	x	x	x	x
33	3	2	5	4	x	x	x	x	x	x	12	μ_2	x	x	x	x	37	v_3	x	x	x	x
34	0	4	4	4	x	x	x	x	x	x	x	x	x	x	x	x	1	w	x	x	x	x
35	1	4	4	4	x	x	x	x	x	x	x	x	x	x	x	x	4	w^*u1	7	w^*u2	9	w^*u3
36	2	4	4	4	x	x	x	x	x	x	x	x	x	x	x	x	10	w^*u1	13	w^*u2	15	w^*u3
37	3	4	4	4	x	x	x	x	x	x	x	x	x	x	x	x	16	w^*u1	19	w^*u2	21	w^*u3

$u1=i_1, u2=i_2, u3=i_3, \mathcal{X}$: current state, $\mathcal{X}'$: next state

Figure 2.2: Transitions and transition rates of the tasking process

3 Design of supervisory control policy

Several possible supervisory control policies associated with tasking are examined. An aggressive policy allocates as many available sensor-pairs to as many targets present; A greedy policy allocates all available sensor-pairs to one target at a time; A conservative policy always allocates only one sensor-pair to every target present to reserve assets in anticipation of new targets. In addition, four optimal policies have been attempted to minimize the cost of sensor loss, threat level, unattended targets, and time needed to replenish upon network expiration, respectively.

The optimal policies are obtained by solving Markov decision problems of appropriate penalty functions. A discrete-time Markov chain model suitable for this purpose can be derived under each cost criterion by the application of a uniformization procedure [9]

$$\pi(t_{k+1}) = \pi(t_k)[I + \frac{1}{\rho}Q(u(x_k))], \quad (3.1)$$

where the uniform rate ρ is greater than any total outgoing transition rates at any states of the original continuous-time Markov chain (2.5).

Each Markov decision problem considered in this report assumes that a cost, denoted by $C(i, u)$, is incurred at every state transition, where i is the state entered and u is a control action selected from a set of admissible actions [4], [2]. A solution amounts to determining a stationary policy $\pi = \{u(x_k), k = 0, 1, \dots\}$ that minimizes the following expected total discounted cost

$$V_\pi(x_0) = E_\pi \sum_{k=0}^{\infty} \alpha^k C(X_k, u_k) \quad (3.2)$$

where $0 < \alpha < 1$ is a discount factor. $C(X_k, u_k)$ in each of the four Markov decision problems takes the form of total number of lost sensor-pairs, ω^{-1} at the state of network expiration, ν_i at the state where it is the server loss, and Q at the state where it is the queue length, respectively.

Let $X_k \in \{1, 2, \dots, 37\}$ denote the random state variable at $t_k = k/\rho$ in the discrete time Markov chain. Control action

$$u(x_k) = \begin{cases} 1, & \text{allocate one sensor-pair to a target} \\ 2, & \text{allocate two sensor-pairs to a target} \\ 3, & \text{allocate three sensor-pairs to a target} \end{cases} \quad (3.3)$$

Note that the indicator functions in Table 1 are defined follows

$$I_i = \begin{cases} 1, & u = i \\ 0, & \text{otherwise} \end{cases}, i = 1, 2, 3. \quad (3.4)$$

It is known [4, 2] that under the condition $0 \leq C(j, u) < \infty$ for all j and all u that belongs to some finite admissible sets U_j , the minimum cost $V^*(i)$ satisfies the following optimality equation:

$$V(i) = \min_{u \in U_i} \left\{ C(i, u) + \alpha \sum_{j=1}^{37} p_{i,j} V(j) \right\}, \quad u \in U_i, \quad (3.5)$$

$i = 1, \dots, 37$, where $p_{i,j}$ is the $(i, j)^{th}$ entry of $I + \frac{1}{\rho} Q(u(x_k))$.

The solution to (3.5) can be obtained via linear programming [3, 2]. In this case, the set of optimality equations is turned into a set of affine constraints on the set of optimization variables $\{V(i)\}$, and the problem can be formally stated as follows.

$$\text{Maximize} \quad V(1) + V(2) + \dots + V(36) + V(37) \quad (3.6)$$

$$\text{Subject to} \quad V(i) \geq 0, \quad i \in \mathcal{X} = \{1, \dots, 37\} \quad (3.7)$$

$$V(i) \leq [C(i, u) + \alpha \sum_j p_{i,j} V(j)]|_u, \quad (3.8)$$

$\forall u \in U_i, \quad i \in \mathcal{X}$.

In the tasking process considered, U_j is nonempty only at state $j = 1, 2, 4, 7, 13, 14, 15, 19, 20, 21, 22, 24, 25, 28, 31, 35, 36, 37$. Therefore, (3.8) leads to 99 affine inequality constraints. This problem is readily solvable by `linprog` in MATLAB's Optimization Toolbox [11]. The active constraints are checked with a MATLAB script to determine the optimal control policy.

Figure 3.1 shows an example of 4 stationary control policies depicted in terms of indicator functions, as defined in (3.4), of the state $x \in \mathcal{X}$. It can be seen that the optimal policy takes into consideration of anticipated targets more than the aggressive policy, but is much more aggressive in terms of use of resources than the conservative policy. Among the four optimal policies solved, only the policy derived under the least sensor loss is plotted (bottom), which will be shown shortly to outperform other three optimal policies in terms of both MTTNE and availability. The minimum queue length policy coincides with the greedy policy, as expected. The other two optimal policies make less aggressive use of resources than the optimal policy shown. The least sensor loss policy will be called the optimal policy from this point on.

The control policies are robust with respect to the range of parameter variations that have been examined: $\nu_1 \in [0.001, 0.01]$ 1/sec., and $\lambda \in [0.001, 0.1]$ 1/sec. The optimal policy is calculated at $\alpha = 0.0792$. All optimal policies drift slightly toward more conservative actions (using fewer resources) as the discount factor α increases, which is consistent with the outcome of the longer term policy making. Because of the finite target population setup, the effect of increasing the target arrival rate is not fully reflective of the target traffic intensity. Simulations with *MATLAB SimEverts* [10] are being performed without limiting the target population.

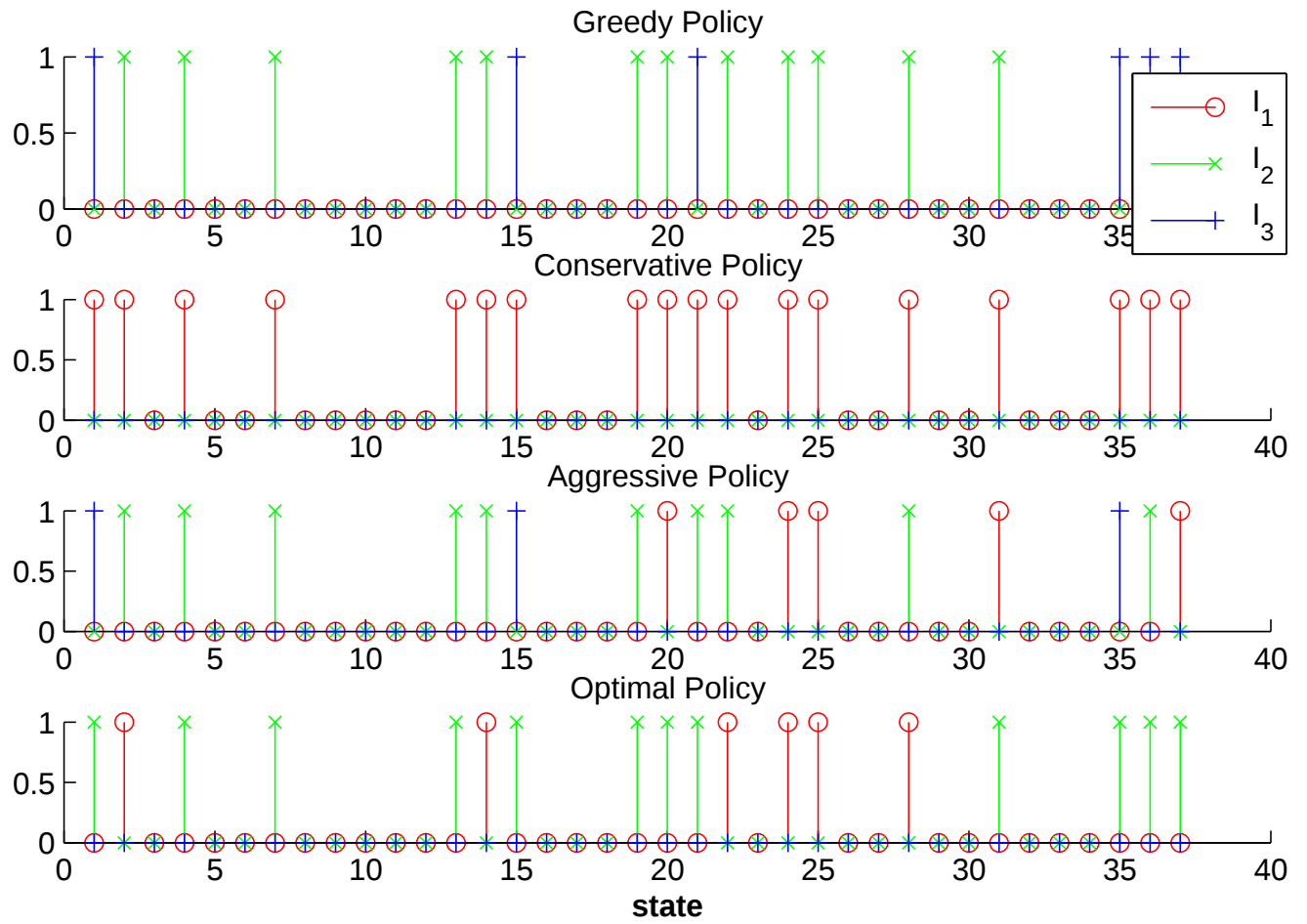


Figure 3.1: Control policy indicators. red: one sever/target; green: two servers/target; blue: three servers/target

4 Performance analysis

The seven policies developed in Section 3 are compared against one another with respect to two common measures of fault-tolerance: mean time to network expiration (MMTNE) and availability. These have been used in [15] in a similar fashion as performance measures of a database unit.

When no replenishment is provided, the network life eventually expires when all sensor-pairs are lost. This occurs when the network enters one of its absorbing states at 34, 35, 36, or 37. Decompose the state probability vector

$$\pi(t) = \left[\underbrace{\pi_\tau(t)}_{1 \times 33} \quad \underbrace{\pi_\alpha(t)}_{1 \times 4} \right] \quad (4.1)$$

where vector $\pi_\tau(t)$ contains transient state probabilities, and $\pi_\alpha(t)$ contains absorbing state probabilities. Decomposing the rate transition matrix Q accordingly yields

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ 0 & 0 \end{bmatrix} \quad (4.2)$$

From (4.2), it can be determined that mean time to network expiration is given by

$$\text{MTTNE} = -\pi_\tau(0)Q_{11}^{-1}1_\tau, \quad 1_\tau = \underbrace{[1 \cdots 1]}_{1 \times 33}^T \quad (4.3)$$

Suppose as soon as the network expires, a replenishment process starts. Suppose with a rate ω the airborne sensors are replenished, and at the completion of the replenishment, the tasking process immediately resumes. In this case, the Markov chain (2.5) becomes irreducible, and a unique steady-state distribution exists [9]. The steady-state availability, which can be roughly thought of as the fraction of time the network has at least one surviving pair of sensors, is computed by

$$A_{net} = 1 - \pi_F(\infty), \quad (4.4)$$

where $\pi_F(\infty) = \pi_{34}(\infty) + \pi_{35}(\infty) + \pi_{36}(\infty) + \pi_{37}(\infty)$, the sum of state probabilities associated with network expiration, which can be determined by solving

$$\pi(\infty)Q = 0, \text{ and } \sum_{x=1}^{37} \pi_x(\infty) = 1. \quad (4.5)$$

A slightly different notion of availability is also examined, where the network is considered unavailable as long as unattended targets are present. In this case, the network availability is given by

$$A_{tgt} = \sum_{i=1}^{11} \pi_i(\infty) + \pi_{13}(\infty) + \pi_{16}(\infty) + \pi_2(\infty) + \pi_{23}(\infty) + \pi_{24}(\infty) + \pi_{26}(\infty) + \pi_{27}(\infty). \quad (4.6)$$

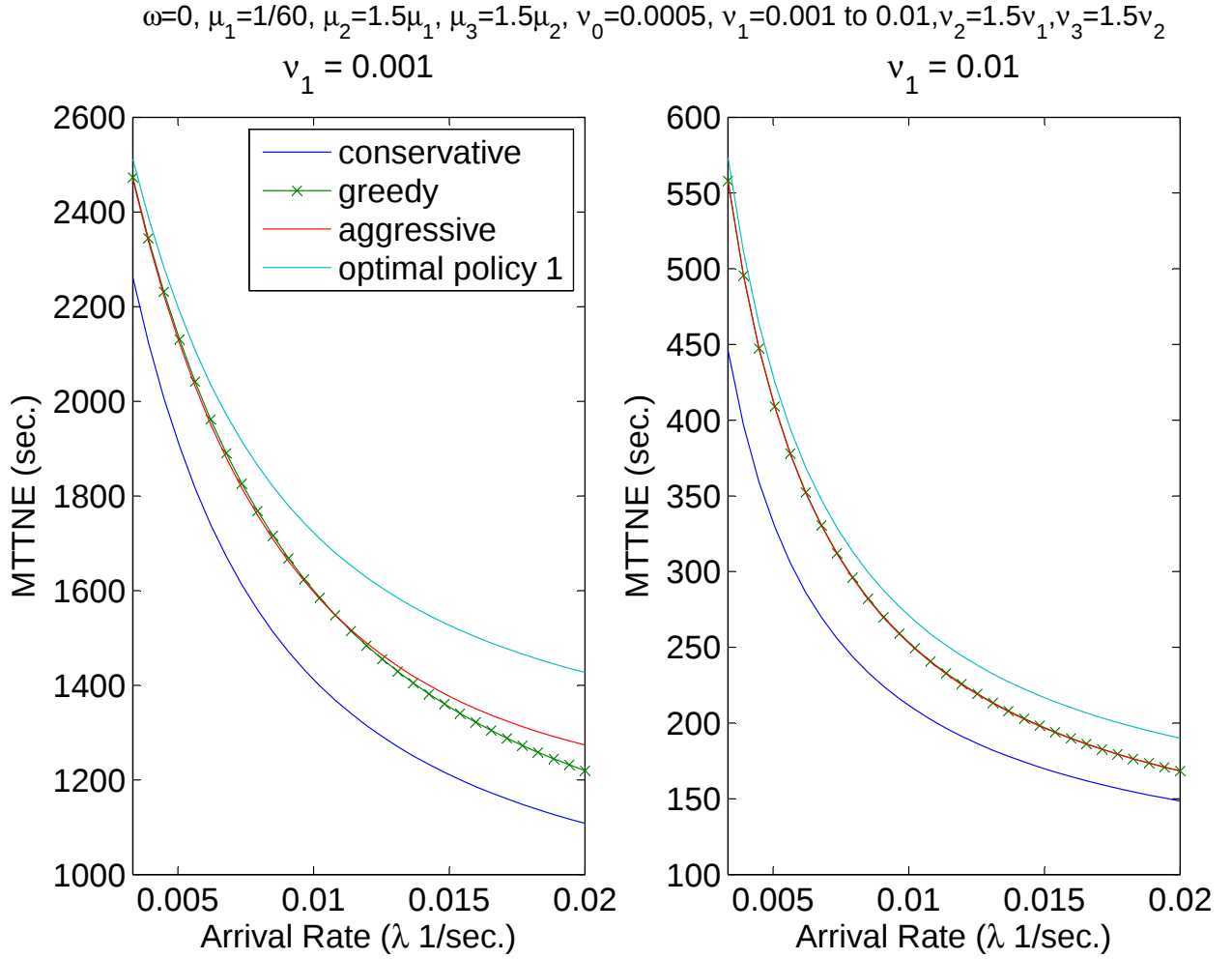


Figure 4.1: Mean time to network expiration as a function of target arrival rate with two sets of sensor loss rates as parameters.

Mean time to network expiration is plotted in Figure 4.1 against target arrival rate with two sets of sensor loss rates as parameters at $\alpha = 0.0792$. It shows that compromise that optimal policy

makes between being too greedy and too conservative enhances the network life consistently for all parameters values considered.

$$\omega=0.0004, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2, v_0=0.0005, v_1=0.001 \text{ to } 0.01, v_2=1.5v_1, v_3=1.5v_2$$

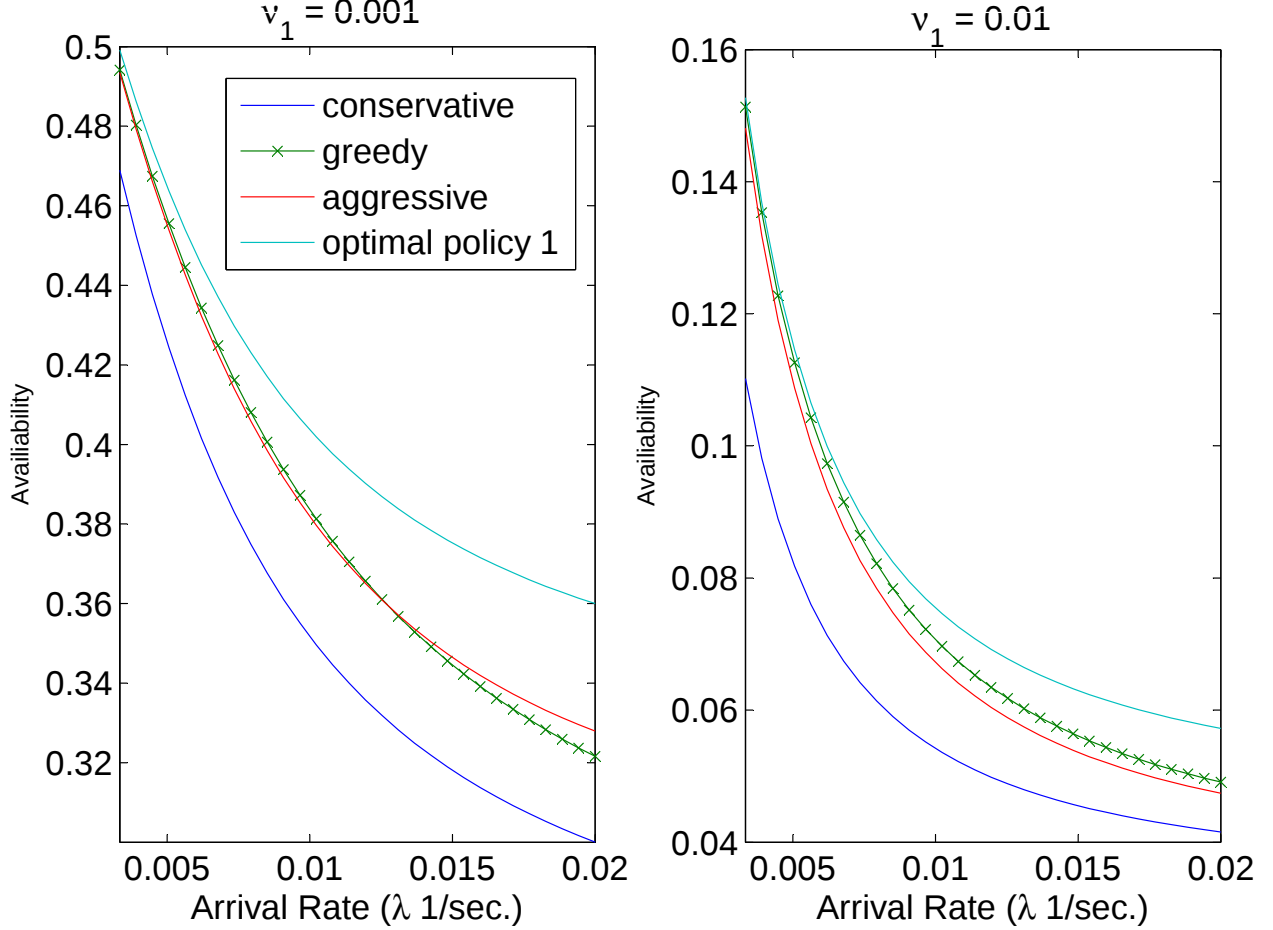


Figure 4.2: Network availability with at least one surviving server.

In Figure 4.2, availability is also plotted against target arrival rate with two sets of sensor loss rates as parameters at discount factor $\alpha = 0.0792$. The observations from the MTTNE apply in terms of the gain the optimal policy offers. It is noted that the availability is low. This is because of the low replenishment rate used in the computation, which corresponds to an expected time of more than 40 minutes to reestablish the lost network.

It is expected that the network availability defined as the probability that all targets is lower than the availability defined as the probability that there is at least one surviving server. On the other hand, the dependence of both notions of availability on the sensor loss rate and on the target arrival rate stays the same. Figure 4.3 shows the plot of the two availabilities against target arrival rate with two sets of sensor loss rates ν_i as parameters under the optimal policy.

Extensive simulations using [10] is being conducted to generate a more complete picture of the network performance in response to control policies for a larger size of networks. The results will be reported elsewhere.

$$\omega=0.0004, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2, v_0=0.0005, \\ v_1=0.001 \text{ to } 0.01, v_2=1.5v_1, v_3=1.5v_2$$

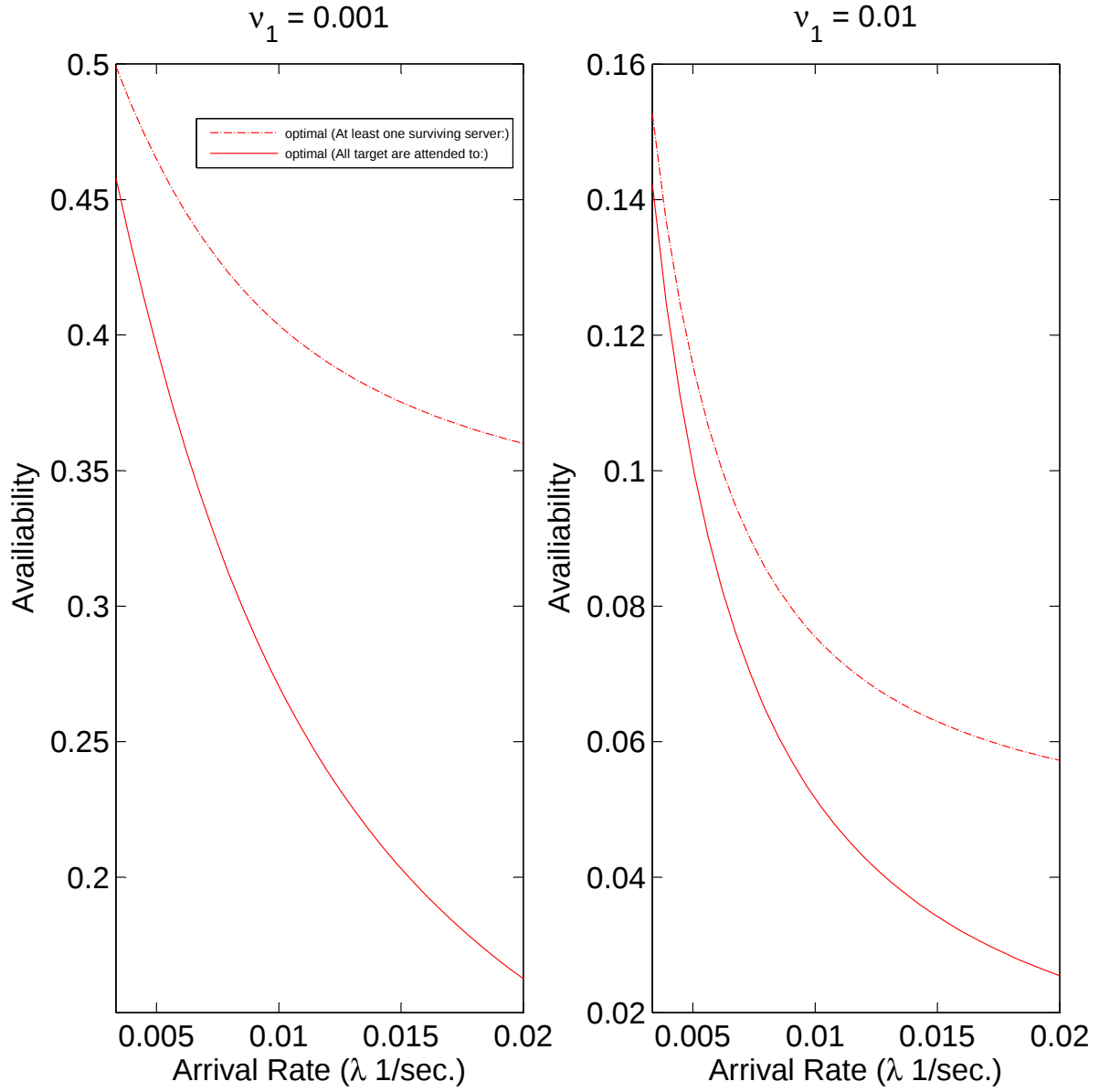


Figure 4.3: Comparison between availability with at least one surviving server (dash) and availability with all targets being attended (solid).

5 Conclusions

This work sought to determine supervisory control policies that best configure an airborne location sensor network to provide a high degree of guarantee of prompt completion of coordinated data acquisition and processing missions in the face of loss of vehicles. Use of redundancy and dynamic allocation of participating sensors was the key enablers.

The report presented a queuing network approach to optimal sensor-pair assignment to locate detected targets for a small scale airborne sensor network, where use of redundancy is balanced with avoiding more vehicle exposure.

A number of related issues are being investigated. In addition to loss of sensors, degradation of network performance can also be the result of broken communication links. Such incidents are modeled as intermittent faults of the servers in queuing networks. The effects of such faults will be studied using discrete event simulations, which will also examine possible emergent phenomenon of larger networks, and non-homogeneous sensors and targets.

A highly relevant task is to solve a guidance and then a control problem of the vehicles. Guidance problem [7] refers to that of establishing a criterion and deriving a set of desired vehicle trajectories under the criterion that the airborne sensors are expected to follow to expedite the target location estimation to a required accuracy. It is known that the quality of acquired data by the airborne sensors depends highly on both the network architecture which is determined by the number of sensor-pairs, and the *states* of all participating sensors relative to the target and to one another. A guidance principle based on the entropy [5] of the noise distribution of the sensed signal has been established, based on which one seeks to adjust the states of the sensors to the most suitable positions and velocities for further data acquisition and processing.

Once the guidance principle is determined, feasible vehicle trajectories can be generated. Path following control can be performed with time coordination [8] to achieve synchronous data acquisition by the sensors. Both the guidance and the control problems are being investigated, and will be reported separately in the near future.

6 Designs and simulation modeling

In general, the sensors of the sensor networks are subject to random breakdowns. This has a heavy influence on the performance measures. Thus, if we model a system containing unreliable sensor-pairs, it is important to take it into account in the model construction. Of course, the breakdown of the sensor-pairs has the most significant negative impact on the performance of the system. The complexity of the system, such as its stochastic, unpredictable behavior, independencies between individual events and the informal nature of many events, makes the analysis profoundly problematic. Such a modeling problem is very significant in large complex systems. The advantage of using simulation is listed below:

1. Simulation provides a controlled environment for performance measurements.
2. Changes can be made easily to the model.
3. Simulation and modeling provides an easy way to control influence of parameters and conduct sensitivity analysis.
4. It does not rely on mathematical concepts heavily and is a very robust technique.

Simulation of the airborne system is conducted in Simulink of MATLAB using the event-based package Simevents [10], [11]. We began with three simulation models. These three models are closed queue with finite homogenous sensor-pairs and finite arrival targets. The first model is reducible, without both system replenishment and channel fading. We used this model to measure the mean time to network expiration (MTTNE). The second model is irreducible with the feature of system replenishment. The third model is also irreducible with both replenishment and channel fading. Both the second and third model is used to measure steady state availability and mean response time of the system. Several control policies associated with tasking are examined in the simulation and compared with the analytic results.

6.1 Basic algorithm of the simulation models

Events do not have equal probability; some events are more likely to occur than others; the execution order of simultaneous events is set to be randomized instead of arbitrary through all simulations. Upon the occurrence of permanent failures, targets are ejected out from the sensor-pairs

before service completion and re-attending the queue with priority, and they require the same service time that was assigned. These targets preempt the waiting targets in the queue but don't preempt the targets that are already in service. Upon occurrence of intermittent failures, targets are ejected out from the sensor-pairs, and they memorize their residual event lives. They immediately return to the assigned sensor-pairs upon recovery of the sensor-pairs. The system is said to be down when all sensor-pairs end their life. No actions can be taken during the repairing process. The event of failure is activated again once the system is operational. The system contains a finite population of targets; the delay elements in the feedback loop generate new targets upon service completion of the targets from one or more sensor-pairs from the upper part of the closed loop system. Appendix C convolutes the simulation and implements the following algorithm:

1. Initialize the system to state 0000.
2. Activate all feasible events
3. Create an initial entity and set a service time for it.
4. Service time is updated according to the number of entities in the system.
5. Failure time of the servers is updated with respect to the number of entities in the system as well. A new failure entity will be generated by the random number generator and preempts the previous failure entity when the queue length changes. The old failure entity is discarded under this condition.
6. Compare the transition times of the feasible events; choose the shortest time and resolve the trigger event. Update to new state upon service completion, entity arrival and server failure.
7. Simulation stops for non-repairable systems once the absorbing states are reached. For repairable systems, simulation stops only if there is a stopping rule.

For most communication systems, redundancy is desired. Assume the stopping rule of this airborne system is defined as the system is down when the backups of servers are also gone or when the system is only allowed to replenish once. Another issue has to do with the failure time. If the current simulation time is greater than the failure time of a sensor-pair, the sensor-pair fails right away. If the current simulation time is less than the failure time, then the service time of the target is equal to the failure time minus the old simulation time.

6.2 Techniques for steady state analysis

Unlike in queuing theory where steady state results in analytic models are easily obtainable, the steady state simulation is not an easy task. Gathering steady state simulation output requires a statistical assertion that the simulation model reached the steady state. The main difficulty is to obtain independent simulation runs with exclusion of the transient period. There are two techniques commonly used for steady state simulation, the Method of Batch means, and the Independent

Replication [18], [19]. None of these two methods is superior to the other in all cases. Intuitively one may say the method of independent replication is superior in producing statistically good estimates for the system's performance measurements. In fact, not one method is superior in all cases, and it all depends on the traffic intensity.

We are using independent replication in this experiment. This technique gets n independent runs of the simulation experiment by running the simulation n times with different initial random seeds for the simulator's random number generator. The main question is determination of the number of runs. The confidence level of simulation output drawn from a set of simulation runs depends on the size of the data set. The larger the number of runs, the higher is the associated confidence. However, more simulation runs require more effort for large systems. Hence, the main goal is to find the smallest number of simulation runs that will also provide the desired confidence. One main limitation of these designs is that the outputs are random from the simulations. Thus, estimates of parameter effects are subject to possibly considerable variance. Unlike physical experiments, we have the luxury in simulation of replicating the runs many times to reduce this variance or perhaps replicating the whole design many times to get many independent and identically distributed estimates of main parameter effects, which could then combine to form a confidence interval. Due to the lengthy simulation time, all results are averaged over 30 simulation runs.

6.3 Performance analysis from simulation

To run a simulation multiple times and gather statistics, the initial seed value would have to be reassigned for each run in all blocks containing it. The MATLAB code initial seed value generator from [10] shown in appendix 9.4 generates random initial seeds for the system. This is one of the methods that may be used to obtain results from multiple runs. To see the performance over the range of arrival rate, we can also use the code 'Running a simulation and varying a parameter' from appendix 9.4. The following subsections are the performance analysis with data collected from simulations. They are plotted in Excel.

6.3.1 Closed - queue reducible model

Fig 6.1 shows the Mean time to network expiration vs. arrival rate. It has the same arrangement as fig. 3.3 with the closeness of the curves at slow arrival rate and the spread of the curves at faster arrival rate. The result is not exactly the same as the analytic result, but very close.

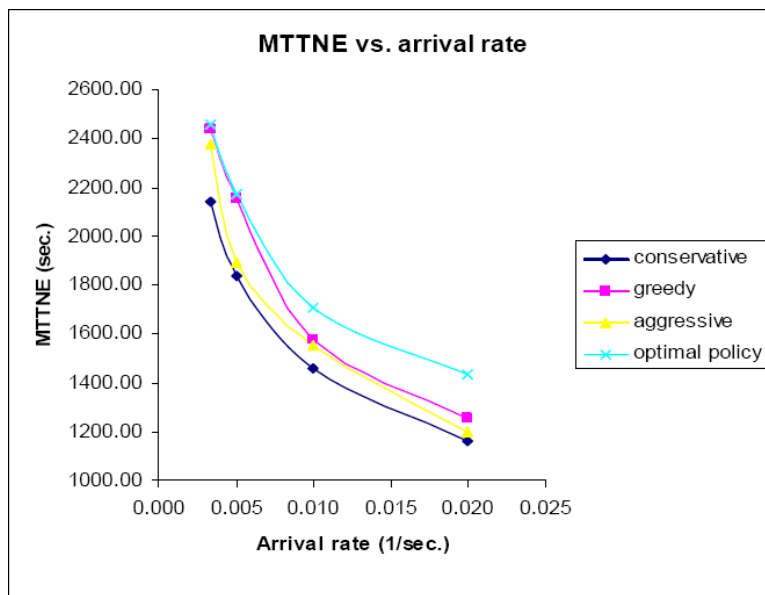


Figure 6.1: Mean time to network expiration vs. arrival rate (simulation model 1)

6.3.2 Closed - queue irreducible model

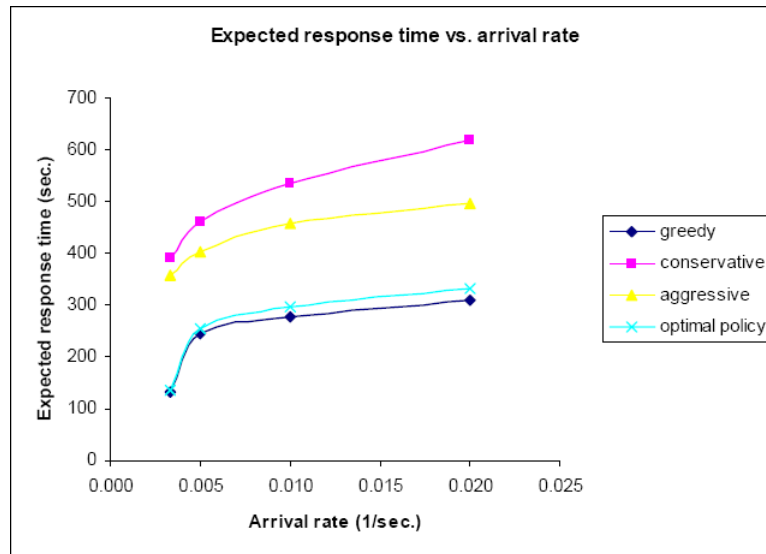


Figure 6.2: Expected response time vs. arrival rate (simulation model 2)

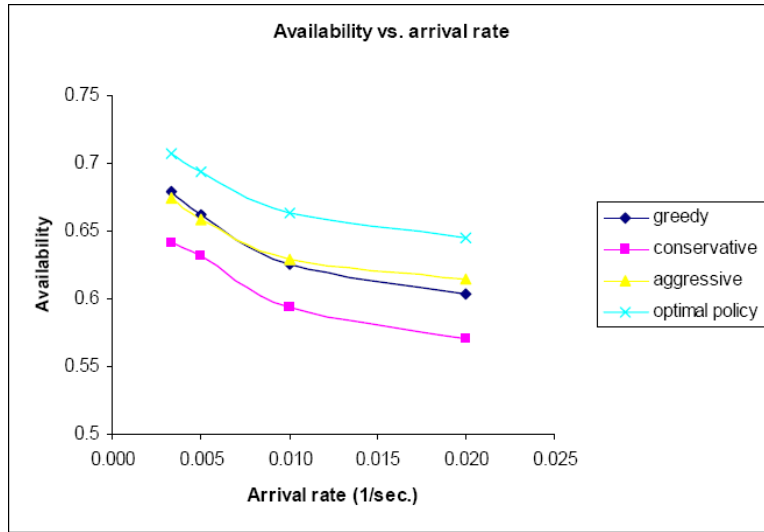


Figure 6.3: Availability vs. arrival rate (simulation model 2)

Compare fig. 6.3 to fig 3.4, the system availability obtain from the simulation is much higher than the analytic model, that is because the simulation only allows to replenish the system once.

6.3.3 Closed - queue irreducible model with intermittent channel fading

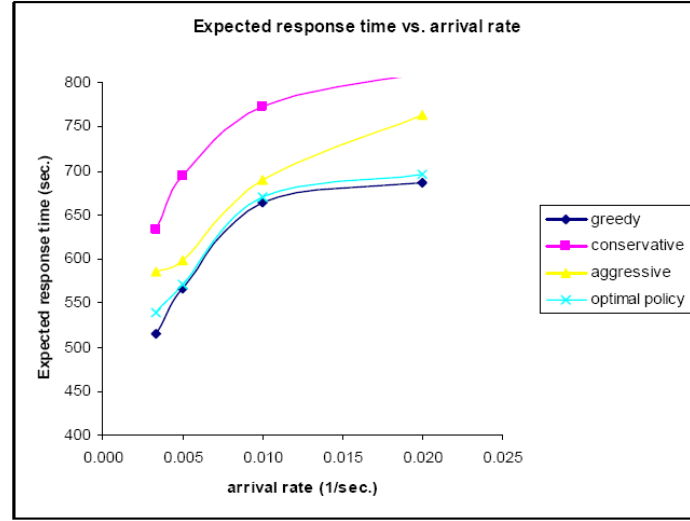


Figure 6.4: Expected response time vs. arrival rate (simulation model 3)

From fig. 6.4, the response time for the system with intermittent channel fading is longer than the response time without intermittent channel fading. It is expected that the time to locate a target would be extended.

6.4 Simulation conclusion

The set up of the simulation models is somewhat different from the analytic model. First of all, in the analytic model, the performance measurements are obtained under all possible states of the system. In the simulation models, the simulation is running under the selection of the control policies, not all system states would be presented. For example, if the conservative policy is applied, a state in which all three sensor-pairs tie together to serve a target would never occur.

Secondly, change of simulation results may be due to the run time of a model and randomness of the event lives. Compare fig. 6.1 to 3.3, the MTTNE is slightly different from the Markov model. The expected response times obtained from simulation model 3 are longer than those for model 2. This is predictable where the intermittent channel fading extends the time to locate targets (additional time is needed for data recovery). The response times from the simulation models 2 and 3 are not the same as fig. 3.6b because fig 3.6b is plotted under the assumption that the number of served targets for all policies is constant. The number of served targets in simulation models is correct and changes by the arrival and service rate. Also, another fact is that the run time for simulations was too short; the systems are not under steady state condition. A more ideal set of data can be obtained by having longer run time with a larger number of departed targets over time.

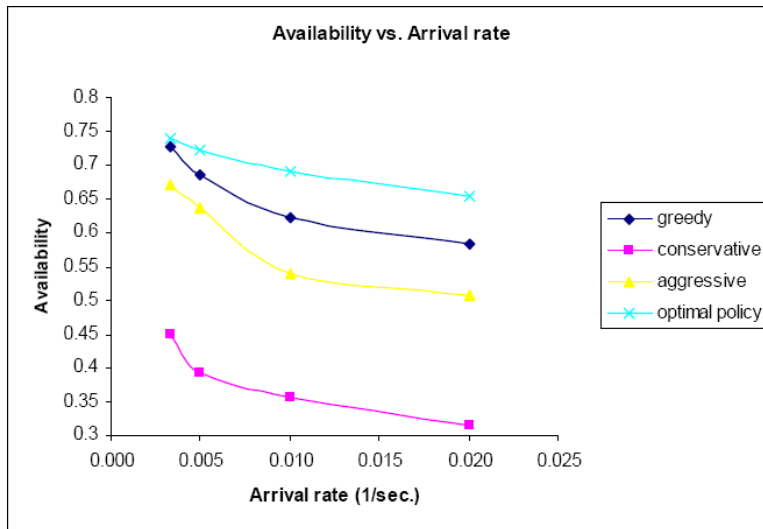


Figure 6.5: Availability vs. arrival rate (simulation model 3)

Similarly, the availabilities from the simulation models are higher than the analytic model. This is also due to the short simulation run time and the effect of non-steady state operation.

7 References

- [1] Antsaklis, P, and Baillieul, J., editors, Special issue on Technology of Networked Control Systems, *Proceedings of the IEEE*, vol.95, Issue 1, 2007.
- [2] Bertsekas, D.P., *Dynamic Programming and Optimal Control*, Volume 1 & Volume 2, Athena Scientific, 1995.
- [3] Boyd, S.P., and Vandenberghe, L., *Convex Optimization*, Cambridge University Press, 2004.
- [4] Cassandras, C.G., and Lafortune, S., *Introduction to Discrete Event Systems*, Kluwer, 1999.
- [5] Cover, T.M., and Thomas, J.A., *Elements of Information Theory*, John Wiley & Sons, 1991.
- [6] Ho, K.C., and Chan, Y.T., Geolocation of a known altitude object from TDOA and FDOA measurements, *IEEE Transactions on Aerospace and Electronic Systems*, vol.33, pp.770-782, 1997.
- [7] Huang, K., Wu, N.E., and Folwer, M.L., Optimal Guidance of Unmanned Aerial Vehicles for Emitter Location, submitted to *2008 American Control Conference*.
- [8] Kaminer, I., Yakimenko, O., Pascoal, A., and Ghabcheloo, R., Path Generation, Path Following and Coordinated Control for Time-Critical Missions of Multiple UAVs , *Proceedings of American Control Conference*, 2006.
- [9] Kao, E.P.C., *An Introduction to Stochastic Processes*, Duxbury Press, 1997.
- [10] MathWorks, *SimEvents User's Guide, For Use with Simulink*, The MathWorks, Inc., 2006.
- [11] MathWorks, *Optimization Toolbox User's Guide, For Use with MATLAB, Version 3*, The MathWorks, Inc., 2006.
- [12] Oh, S., Scheanto, L., Chen, P., and Sastry, S., Tracking and coordination of multiple agents using sensor networks: system design, algorithms, and experiments, *Proceedings of the IEEE*, vol.95, pp.234-254, 2007.
- [13] Samad, T., Bay, J.S., and Godbole, D., Network-centric systems for military operations in urban terrain: the role of UAVs, *Proceedings of the IEEE*, vol.95, pp.92-107, 2007.

- [14] Torrieri, D.J., Statistical theory of passive location systems, *IEEE Transactions on Aerospace and Electronic Systems*, vol.20, 1984.
- [15] Wu, N.E., Metzler, J.M., Linderman, M.H., Supervisory Control of a Database Unit, *Proc. IEEE Conference on Decision and Control*, 2005.

8 MATLAB code for the analytic model of the airborne network

8.1 Transition rate matrix

The transition rate matrix is computed from the transition rate diagram from the previous page or from Table 4.

8.2 Mean time to network expiration

```
1 clc, clear
2
3 % parameters
4 la=linspace(1/300, 1/50, 30);
5 nu1=linspace(1/1000, 1/100, 2);
6 nu0=0.0005; nu2=1.5*nu1; nu3=1.5*nu2;
7 mu1=1/60; mu2=1.5*mu1; mu3=1.5*mu2;
8
9 %% compute mean time to network expiration (reducible case)
10 for i=1:length(la)
11     for j=1:length(nu1)
12         setpara(mu1, mu2, mu3, nu0, nu1(j), nu2(j), nu3(j), la(i), 0);
13
14         Q =Qnew1;
15
16         pi_T = [1 zeros(1, (length(Q)-5))];
17         ones_T = ones((length(Q)-4), 1);
18         Q_T = Q(1:(length(Q)-4), 1:(length(Q)-4));
19         MTTNE1(i, j) = -pi_T * inv(Q_T) * ones_T; % greedy
20
21         Q2=Qnew2;
22
23         pi_T2 = [1 zeros(1, (length(Q2)-5))];
24         ones_T2 = ones((length(Q2)-4), 1);
25         Q2_T = Q2(1:(length(Q2)-4), 1:(length(Q2)-4));
26         MTTNE2(i, j) = -pi_T2 * inv(Q2_T) * ones_T2; % conservative
```

```

27
28     Qag=Qnewag;
29
30     pi_Tag = [1 zeros(1,(length(Qag)-5))];
31     ones_Tag = ones((length(Qag)-4),1);
32     Qag_T = Qag(1:(length(Qag)-4),1:(length(Qag)-4));
33     MTTNEag(i,j) = -pi_Tag * inv(Qag_T) * ones_Tag;           % aggressive
34
35     Qopt1=Qnewopt1;
36     pi_Topt1 = [1 zeros(1,(length(Qopt1)-5))];
37     ones_Topt1 = ones((length(Qopt1)-4),1);
38     Qopt1_T = Qopt1(1:(length(Qopt1)-4),1:(length(Qopt1)-4));
39     MTTNEopt1(i,j) = -pi_Topt1 * inv(Qopt1_T)
40         * ones_Topt1;    % optimal policy 1
41
42     Qopt2=Qnewopt2;
43     pi_Topt2 = [1 zeros(1,(length(Qopt2)-5))];
44     ones_Topt2 = ones((length(Qopt2)-4),1);
45     Qopt2_T = Qopt2(1:(length(Qopt2)-4),1:(length(Qopt2)-4));
46     MTTNEopt2(i,j) = -pi_Topt2 * inv(Qopt2_T)...
47         * ones_Topt2;    % optimal policy 2
48
49     Qopt3=Qnewopt3;
50     pi_Topt3 = [1 zeros(1,(length(Qopt3)-5))];
51     ones_Topt3 = ones((length(Qopt3)-4),1);
52     Qopt3_T = Qopt3(1:(length(Qopt3)-4),1:(length(Qopt3)-4));
53     MTTNEopt3(i,j) = -pi_Topt3 * inv(Qopt3_T)...
54         * ones_Topt3;    % optimal policy 3
55
56 end,end
57
58 diff1 = MTTNEag-MTTNE1;           % diff between aggressive & greedy
59 diff2 = MTTNE1-MTTNE2;           % diff between greedy & conservative
60 diff3 = MTTNEag - MTTNE2;        % diff between aggressive & conservative
61
62 %% plot MTTNE vs. arrival rate
63
64 figure(1)
65 subplot(1,2,1)
66 plot(la,MTTNE2(:,1), la,MTTNE1(:,1), '-x', la,MTTNEag(:,1), la,MTTNEopt1(:,1))
67 title({'\omega=0, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2, \nu_0=0.0005,...'
68     '\nu_1=0.001 to 0.01, \nu_2=1.5\nu_1, \nu_3=1.5\nu_2';...
69     'MEAN TIME TO NETWORK EXPIRATION VS. ARRIVAL RATE';'\nu_1 = 0.001'})
70
71 legend('conservative','greedy','aggressive','optimal policy 1')
72 xlabel('Arrival Rate (\lambda 1/sec.)')
73 ylabel('MTTNE (sec.)', xlim([1/300, 1/50])
74 subplot(1,2,2)
75 plot(la,MTTNE2(:,2), la,MTTNE1(:,2), '-x', la,MTTNEag(:,2), la,MTTNEopt1(:,2))
76 title('\nu_1 = 0.01', xlabel('Arrival Rate (\lambda 1/sec.)')
77 ylabel('MTTNE (sec.)', xlim([1/300, 1/50])

```

```

78
79 figure(2)
80 subplot(1,2,1), plot(la,MTTNEopt1(:,1), '-+', la, MTTNEopt2(:,1), '-o',....
81     la,MTTNEopt3(:,1))
82 title({'\omega=0, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2,...
83     \nu_0=0.0005, \nu_1=0.001 to 0.01, \nu_2=1.5\nu_1, \nu_3=1.5\nu_2';...
84     'MEAN TIME TO NETWORK EXPIRATION VS. ARRIVAL RATE';'\nu_1 = 0.001'})
85
86 legend('optimal policy 1', 'optimal policy 2','optimal policy 3')
87 xlabel('Arrival Rate (\lambda 1/sec.)'), ylabel('MTTNE (sec.)')
88 xlim([1/300, 1/50])
89 subplot(1,2,2)
90 plot(la,MTTNEopt1(:,2), '-+', la, MTTNEopt2(:,2), '-o', la,MTTNEopt3(:,2))
91 title('\nu_1 = 0.01')
92 xlabel('Arrival Rate (\lambda 1/sec.)')
93 ylabel('MTTNE (sec.)')
94 xlim([1/300, 1/50])

```

8.3 Steady-state availability

```

1 clc,clear
2
3 % parameters
4 la=linspace(1/300, .02, 30);
5 nu1=linspace(1/1000, 1/100, 2);
6 nu0=0.0005; nu2=1.5*nu1; nu3=1.5*nu2;
7 mu1=1/60; mu2=1.5*mu1; mu3=1.5*mu2;
8 w=0.0004;
9
10 %% computing steady state availability
11 for i=1:length(la)
12     for j=1:length(nu1)
13         setpara(mu1, mu2, mu3, nu0, nu1(j), nu2(j), nu3(j), la(i), w);
14
15         Q =Qnew1;           %greedy
16         Q2=Qnew2;           %conservative
17         Qag=Qnewag;         %aggressive
18         Qopt1=Qnewopt1;     %optimal 1
19         Qopt2=Qnewopt2;     %optimal 2
20         Qopt3=Qnewopt3;     %optimal 3
21
22         Q(:,length(Q)) = [ones(1,length(Q))];
23         p_ss = [zeros(1,(length(Q)-1)), 1]*inv(Q);
24
25 %avail 1 when the system is up
26 A_sys1(i,j) = p_ss(1:(length(Q)-4))*ones((length(Q)-4),1);
27
28 %avail 2: all targets are in service
29 A_sys1_2(i,j) = 1-p_ss(12)-p_ss(14)-p_ss(15)-p_ss(17)-p_ss(18)-p_ss(19)...

```

```

30     -p_ss(20)-p_ss(21)-p_ss(25)-p_ss(28)-p_ss(29)-p_ss(30)-p_ss(31) ...
31     -p_ss(32)-p_ss(33)-p_ss(34)-p_ss(35)-p_ss(36)-p_ss(37);
32
33     Q2(:,length(Q2)) = [ones(1,length(Q2))];
34     p_ss2 = [zeros(1,(length(Q2)-1)), 1]*inv(Q2);
35     A_sys2(i,j) = p_ss2(1:(length(Q2)-4))*ones((length(Q2)-4),1);
36     A_sys2_2(i,j) = 1-p_ss2(12)-p_ss2(14)-p_ss2(15)-p_ss2(17)-p_ss2(18) ...
37     -p_ss2(19)-p_ss2(20)-p_ss2(21)-p_ss2(25)-p_ss2(28)-p_ss2(29) ...
38     -p_ss2(30)-p_ss2(31)-p_ss2(32)-p_ss2(33)-p_ss2(34)-p_ss2(35) ...
39     -p_ss2(36)-p_ss2(37);
40
41     Qag(:,length(Qag)) = [ones(1,length(Qag))];
42     p_ssag = [zeros(1,(length(Qag)-1)), 1]*inv(Qag);
43     A_sysag(i,j) = p_ssag(1:(length(Qag)-4))*ones((length(Qag)-4),1);
44     A_sysag_2(i,j) = 1-p_ssag(12)-p_ssag(14)-p_ssag(15)-p_ssag(17)-p_ssag(18) ...
45     -p_ssag(19)-p_ssag(20)-p_ssag(21)-p_ssag(25)-p_ssag(28)-p_ssag(29) ...
46     -p_ssag(30)-p_ssag(31)-p_ssag(32)-p_ssag(33)-p_ssag(34)-p_ssag(35) ...
47     -p_ssag(36)-p_ssag(37);
48
49     Qopt1(:,length(Qopt1)) = [ones(1,length(Qopt1))];
50     p_ssopt1 = [zeros(1,(length(Qopt1)-1)), 1]*inv(Qopt1);
51     A_sysopt1(i,j) = p_ssopt1(1:(length(Qopt1)-4)) ...
52     *ones((length(Qopt1)-4),1);
53     A_sysopt1_2(i,j) = 1-p_ssopt1(12)-p_ssopt1(14)-p_ssopt1(15)-p_ssopt1(17) ...
54     -p_ssopt1(18)-p_ssopt1(19)-p_ssopt1(20)-p_ssopt1(21)-p_ssopt1(25) ...
55     -p_ssopt1(28)-p_ssopt1(29)-p_ssopt1(30)-p_ssopt1(31)-p_ssopt1(32) ...
56     -p_ssopt1(33)-p_ssopt1(34)-p_ssopt1(35)-p_ssopt1(36)-p_ssopt1(37);
57
58     Qopt2(:,length(Qopt2)) = [ones(1,length(Qopt2))];
59     p_ssopt2 = [zeros(1,(length(Qopt2)-1)), 1]*inv(Qopt2);
60     A_sysopt2(i,j) = p_ssopt2(1:(length(Qopt2)-4)) ...
61     *ones((length(Qopt2)-4),1);
62     A_sysopt2_2(i,j) = 1-p_ssopt2(12)-p_ssopt2(14)-p_ssopt2(15)-p_ssopt2(17) ...
63     -p_ssopt2(18)-p_ssopt2(19)-p_ssopt2(20)-p_ssopt2(21)-p_ssopt2(25) ...
64     -p_ssopt2(28)-p_ssopt2(29)-p_ssopt2(30)-p_ssopt2(31)-p_ssopt2(32) ...
65     -p_ssopt2(33)-p_ssopt2(34)-p_ssopt2(35)-p_ssopt2(36)-p_ssopt2(37);
66
67
68     Qopt3(:,length(Qopt3)) = [ones(1,length(Qopt3))];
69     p_ssopt3 = [zeros(1,(length(Qopt3)-1)), 1]*inv(Qopt3);
70     A_sysopt3(i,j) = p_ssopt3(1:(length(Qopt3)-4)) ...
71     *ones((length(Qopt3)-4),1);
72     A_sysopt3_2(i,j) = 1-p_ssopt3(12)-p_ssopt3(14)-p_ssopt3(15)-p_ssopt3(17) ...
73     -p_ssopt3(18)-p_ssopt3(19)-p_ssopt3(20)-p_ssopt3(21)-p_ssopt3(25) ...
74     -p_ssopt3(28)-p_ssopt3(29)-p_ssopt3(30)-p_ssopt3(31)-p_ssopt3(32) ...
75     -p_ssopt3(33)-p_ssopt3(34)-p_ssopt3(35)-p_ssopt3(36)-p_ssopt3(37);
76
77     end,end
78
79     diff1=A_sysag - A_sys1;           % diff between aggressive and greedy
80     diff2=A_sys1 - A_sys2;           % diff between greedy and conservative

```

```

81 diff3=A_sysag-A_sys2;           % diff between aggressive and conservative
82
83 %% plots: avail 1 vs. arrival rate
84
85 figure(1)
86 subplot(1,2,1)
87 plot(la,A_sys2(:,1), la,A_sys1(:,1), '-x', la,A_sysag(:,1), la, A_sysopt1(:,1))
88 title({'\omega=0.0004, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2,...
89         \nu_0=0.0005, \nu_1=0.001 to 0.01, \nu_2=1.5\nu_1, \nu_3=1.5\nu_2';...
90         'AVAILABILITY VS. ARRIVAL RATE';'\nu_1 = 0.001'})
91 legend('conservative','greedy','aggressive','optimal policy 1')
92 xlabel('Arrival Rate (\lambda 1/sec.)'),ylabel('Availiability')
93 xlim([1/300, 1/50])
94 subplot(1,2,2)
95 plot(la,A_sys2(:,2), la,A_sys1(:,2), '-x', la,A_sysag(:,2), la, A_sysopt1(:,2))
96 title('\nu_1 = 0.01'), xlabel('Arrival Rate (\lambda 1/sec.)')
97 ylabel('Availiability'), xlim([1/300, 1/50])
98
99 figure(2)
100 subplot(1,2,1), plot(la,A_sysopt1(:,1), la,A_sysopt2(:,1), '-+', la,A_sysopt3(:,1))
101 title({'\omega=0.0004, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2,...
102         \nu_0=0.0005, \nu_1=0.001 to 0.01, \nu_2=1.5\nu_1, \nu_3=1.5\nu_2';...
103         'AVAILABILITY VS. ARRIVAL RATE';'\nu_1 = 0.001'})
104 legend('optimal policy 1', 'optimal policy 2','optimal policy 3')
105 xlabel('Arrival Rate (\lambda 1/sec.)'), ylabel('Availiability')
106 xlim([1/300, 1/50])
107 subplot(1,2,2), plot(la,A_sysopt1(:,2), la,A_sysopt2(:,2), '-+', la,A_sysopt3(:,2))
108 title('\nu_1 = 0.01'), xlabel('Arrival Rate (\lambda 1/sec.)')
109 ylabel('Availiability'), xlim([1/300, 1/50])
110
111 %% plot of avail 1 vs. avail 2
112 figure(3)
113 subplot(1,2,1)
114 plot(la,A_sysopt1(:,1), '-.r', la,A_sysopt2(:,1), '-xg',...
115         la,A_sysopt3(:,1), '-ob', la,A_sysopt1_2(:,1), 'r', la,...
116         A_sysopt2_2(:,1), 'g', la,A_sysopt3_2(:,1), 'b')
117 title({'\omega=0.0004, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2,...
118         \nu_0=0.0005, \nu_1=0.001 to 0.01, \nu_2=1.5\nu_1, \nu_3=1.5\nu_2';...
119         'AVAILABILITY VS. ARRIVAL RATE';'\nu_1 = 0.001'})
120 legend('optimal 1 (avail 1)', 'optimal 2 (avail 1)','optimal 3 (avail 1)',...
121         'optimal 1 (avail 2)', 'optimal 2 (avail 2)', 'optimal 3 (avail 2)')
122 xlabel('Arrival Rate (\lambda 1/sec.)'), ylabel('Availiability')
123 xlim([1/300, 1/50])
124 subplot(1,2,2)
125 plot(la,A_sysopt1(:,2), '-.r', la,A_sysopt2(:,2), '-xg', la,A_sysopt3(:,2),...
126         '-ob', la,A_sysopt1_2(:,2), 'r', la,A_sysopt2_2(:,2), 'g', la,...
127         A_sysopt3_2(:,2), 'b')

```

8.4 Expected response time

```

1      state = [ 0      0      0      0
2                0      4      0      0
3                0      4      4      0
4                1      1      0      0
5                1      1      4      0
6                1      1      4      4
7                1      2      2      0
8                1      2      2      4
9                1      3      3      3
10               2      1      1      0
11               2      1      1      4
12               2      1      4      4
13               2      2      2      1
14               2      2      2      4
15               2      3      3      3
16               3      1      1      1
17               3      1      1      4
18               3      1      4      4
19               3      2      2      1
20               3      2      2      4
21               3      3      3      3
22               1      2      5      0
23               1      2      5      4
24               2      2      5      1
25               3      2      5      1
26               1      3      3      5
27               1      3      5      5
28               2      3      3      5
29               2      3      5      5
30               2      2      5      4
31               3      3      3      5
32               3      3      5      5
33               3      2      5      4
34               0      4      4      4
35               1      4      4      4
36               2      4      4      4
37               3      4      4      4];
38
39 %% parameters
40
41 la=linspace(1/300, 0.02, 30);
42 nu1=linspace(1/1000, 1/100, 2);
43 nu0=0.0005; nu2=1.5*nu1; nu3=1.5*nu2;
44 mu1=1/60; mu2=1.5*mu1; mu3=1.5*mu2;
45 w=0.0004;
46
47 for i=1:length(la)
48     for j=1:length(nu1)
49         setpara(mu1, mu2, mu3, nu0, nu1(j), nu2(j), nu3(j), la(i), w);
50         failure = [nu0 nu0 nu0 nu1 nu1 nu1 nu1 nu1 nu1 nu2 nu2 nu2 nu2 nu2...
51                   nu2 nu3 nu3 nu3 nu3 nu3 nu3 nu1 nu1 nu2 nu3 nu1 nu1 nu2 nu2 nu2...

```

```

52         nu3 nu3 nu3 0 0 0 0];
53
54         Q =Qnew1;                                % greedy case
55         Q(:,length(Q)) = [ones(1,length(Q))];
56         p_ss = [zeros(1,(length(Q)-1)), 1]*inv(Q);
57         T=p_ss*state(:,1);                        % expected number of targets
58         R(i,j)=T/((3-T)*la(i));                  % expected response time
59
60         Q2=Qnew2;                                % conservative case
61         Q2(:,length(Q2)) = [ones(1,length(Q2))];
62         p_ss2 = [zeros(1,(length(Q2)-1)), 1]*inv(Q2);
63         T2=p_ss2*state(:,1);                     % expected number of targets
64         R2(i,j)=T2/((3-T2)*la(i));               % expected response time
65
66         Qag=Qnewag;                               % aggressive case
67         Qag(:,length(Qag)) = [ones(1,length(Qag))];
68         p_ssag = [zeros(1,(length(Qag)-1)), 1]*inv(Qag);
69         Tag=p_ssag*state(:,1);                   % expected number of targets
70         Rag(i,j)=Tag/((3-Tag)*la(i));            % expected response time
71
72         Qopt1=Qnewopt1;                           % optimal 1
73         Qopt1(:,length(Qopt1)) = [ones(1,length(Qopt1))];
74         p_ss_opt1 = [zeros(1,(length(Qopt1)-1)), 1]*inv(Qopt1);
75         Topt1=p_ss_opt1*state(:,1);              % expected number of targets
76         Ropt1(i,j)=Topt1/((3-Topt1)*la(i));      % expected response time
77
78         Qopt2=Qnewopt2;                           % optimal 2
79         Qopt2(:,length(Qopt2)) = [ones(1,length(Qopt2))];
80         p_ss_opt2 = [zeros(1,(length(Qopt2)-1)), 1]*inv(Qopt2);
81         Topt2=p_ss_opt2*state(:,1);              % expected number of targets
82         Ropt2(i,j)=Topt2/((3-Topt2)*la(i));      % expected response time
83
84         Qopt3=Qnewopt3;                           % optimal 3
85         Qopt3(:,length(Qopt3)) = [ones(1,length(Qopt3))];
86         p_ss_opt3 = [zeros(1,(length(Qopt3)-1)), 1]*inv(Qopt3);
87         Topt3=p_ss_opt3*state(:,1);              % expected number of targets
88         Ropt3(i,j)=Topt3/((3-Topt3)*la(i));      % expected response time
89
90     end,end
91
92 % response time under the breakdown condition of servers
93 % p=10; % assume 10 customers have completed the service.
94 %     R=(R*p+1/w*ones(30,2))/p;
95 %     R2=(R2*p+1/w*ones(30,2))/p;
96 %     Rag=(Rag*p+1/w*ones(30,2))/p;
97 %     Ropt1=(Ropt1*p+1/w*ones(30,2))/p;
98 %     Ropt2=(Ropt2*p+1/w*ones(30,2))/p;
99 %     Ropt3=(Ropt3*p+1/w*ones(30,2))/p;
100
101 %% response time vs. arrival rate
102

```

```

103 figure(1)
104 subplot(1,2,1)
105 plot(la,R2(:,1), '-o', la,R(:,1), '-x', la,Rag(:,1), '--', la,Ropt1(:,1),...
106      '-.', la,Ropt2(:,1), '-+', la, Ropt3(:,1))
107 title({'\omega=0.0004, \mu_1=1/60, \mu_2=1.5\mu_1, \mu_3=1.5\mu_2,...
108      \nu_0=0.0005, \nu_1=0.001 to 0.01, \nu_2=1.5\nu_1, \nu_3=1.5\nu_2';...
109      'RESPONSE TIME VS. FAILURE RATE';'\nu_1 = 0.001'})
110 ylabel('Response Time (sec.)', xlim([1/300, 1/50])
111
112 subplot(1,2,2)
113 plot(la,R2(:,2), '-o', la,R(:,2), '-x', la,Rag(:,2), '--', la,Ropt1(:,2),...
114      '-.', la,Ropt2(:,2), '-+', la, Ropt3(:,2))
115 legend('conservative','greedy=optimal 4','aggressive','optimal 1',...
116      'optimal 2','optimal 3')
117 title('\nu_1 = 0.02'), xlabel('Arrival Rate (\lambda 1/sec.)')
118 ylabel('Response Time (sec.)', xlim([1/300, 1/50])

```

8.5 Linear Programming

```

1  for k = 1:length(state)
2      l(k) = state(k,1);           %queue length at each state
3  end
4  %% parameters
5  la=1/100;           % arrival rate
6  w=0.0004;           % overhaul rate
7  nu1=1/1000; nu0=0.0005; nu2=1.5*nu1; nu3=1.5*nu2;           %failure rate
8  mu1=1/60; mu2=1.5*mu1; mu3=1.5*mu2;           %service rate
9  rho = 3*(mu1+mu2+mu3+la+nu1+nu2+nu3)+w;           %uniform rate
10 beta = 3;           % discount factor
11 alpha = rho/(beta+rho);           % discount factor for equivalent Discrete Markov
12
13 % number of failed servers in each state
14 m = [0 1 2 0 1 2 0 1 0 0 1 2 0 1 0 0 1 2 0 1 0 1 2 1 1 1 2 1 2 2 1 2 2 3 3 3 3];
15 %states would lead to system failure
16 n = [0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 0 0 0 0];
17 p = [3*nu0 2*nu0 nu0 3*nu1 2*nu1 nu1 2*nu1 2*nu1 3*nu1 3*nu2...
18      2*nu2 nu2 2*nu2 2*nu2 3*nu2 3*nu3 2*nu3 nu3 2*nu3 2*nu3...
19      3*nu3 nu1 nu1 nu2 nu3 2*nu1 nu1 2*nu2 nu2 nu2 2*nu3 nu3 nu3 0 0 0 0];
20
21 step_cost = m;           %cost of failed servers
22 % step_cost = n/w ; %cost of replenish
23 % step_cost = p;           %cost of failure rate
24 % step_cost = l;           %cost of queue length
25
26 setpara(mu1, mu2, mu3, nu0, nu1, nu2, nu3, la, w);
27 Q1 = Qmatrix(1,0,0);
28 Q2 = Qmatrix(0,1,0);
29 Q3 = Qmatrix(0,0,1);
30 P1=eye(37)+Q1/rho; P2=eye(37)+Q2/rho; P3=eye(37)+Q3/rho; %P matrices

```

```

31
32 %% linear programming
33 f = -ones(37,1); %length of f must be same as number of columns of A.
34
35 A = -alpha*[P1-eye(37)/alpha; P2-eye(37)/alpha;
36     P3(1,1)-1/alpha P3(1,2:37);
37     P3(3,1:2) P3(3,3)-1/alpha P3(3,4:37);
38     P3(5,1:4) P3(5,5)-1/alpha P3(5,6:37);
39     P3(6,1:5) P3(6,6)-1/alpha P3(6,7:37);
40     P3(8,1:7) P3(8,8)-1/alpha P3(8,9:37);
41     P3(9,1:8) P3(9,9)-1/alpha P3(9,10:37);
42     P3(10,1:9) P3(10,10)-1/alpha P3(10,11:37);
43     P3(11,1:10) P3(11,11)-1/alpha P3(11,12:37);
44     P3(12,1:11) P3(12,12)-1/alpha P3(12,13:37);
45     P3(15,1:14) P3(15,15)-1/alpha P3(15,16:37);
46     P3(16,1:15) P3(16,16)-1/alpha P3(16,17:37);
47     P3(17,1:16) P3(17,17)-1/alpha P3(17,18:37);
48     P3(18,1:17) P3(18,18)-1/alpha P3(18,19:37);
49     P3(21,1:20) P3(21,21)-1/alpha P3(21,22:37);
50     P3(23,1:22) P3(23,23)-1/alpha P3(23,24:37);
51     P3(26,1:25) P3(26,26)-1/alpha P3(26,27:37);
52     P3(27,1:26) P3(27,27)-1/alpha P3(27,28:37);
53     P3(29,1:28) P3(29,29)-1/alpha P3(29,30:37);
54     P3(30,1:29) P3(30,30)-1/alpha P3(30,31:37);
55     P3(32,1:31) P3(32,32)-1/alpha P3(32,33:37);
56     P3(33,1:32) P3(33,33)-1/alpha P3(33,34:37);
57     P3(34,1:33) P3(34,34)-1/alpha P3(34,35:37);
58     P3(35,1:34) P3(35,35)-1/alpha P3(35,36:37);
59     P3(36,1:35) P3(36,36)-1/alpha P3(36,37);
60     P3(37,1:36) P3(37,37)-1/alpha];
61
62 b =[step_cost step_cost step_cost(1) step_cost(3) step_cost(5)...
63     step_cost(6) step_cost(8) step_cost(9) step_cost(10) step_cost(11)...
64     step_cost(12) step_cost(15) step_cost(16) step_cost(17) step_cost(18)...
65     step_cost(21) step_cost(23) step_cost(26) step_cost(27) step_cost(29)...
66     step_cost(30) step_cost(32) step_cost(33) step_cost(34) step_cost(35)...
67     step_cost(36) step_cost(37)]'; %length b must be = number of rows of A.
68
69 lb = zeros(37,1);
70 [x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb);
71 %% comparison
72 for i=1:37
73     v1(i)=step_cost(i)+alpha*(P1(i,:)*x);
74     v2(i)=step_cost(i)+alpha*(P2(i,:)*x);
75     v3(i)=step_cost(i)+alpha*(P3(i,:)*x);
76     a1(i)=v1(i)-x(i);
77     a2(i)=v2(i)-x(i);
78     a3(i)=v3(i)-x(i);
79 end
80
81 %display optimal cost and cost from the three control sets

```

```

82 disp('  opt_value      v1      v2      v3')
83 disp([x  v1' v2' v3'])
84
85 % determine the control policy for each state by
86 % checking out the smallest value among a1 a2 a3
87 disp('control policy: ')
88 disp('u1  u2  u3')
89 for i=1:37
90     if (abs(a1(i)') > abs(a3(i)')) & (abs(a2(i)') > abs(a3(i)'))
91         disp(' 0      0      1')
92     elseif (abs(a1(i)') > abs(a2(i)')) & (abs(a3(i)') > abs(a2(i)'))
93         disp(' 0      1      0')
94     elseif (abs(a2(i)') > abs(a1(i)')) & (abs(a3(i)') > abs(a1(i)'))
95         disp(' 1      0      0')
96     else disp(' 0      0      0')
97     end
98 end

```

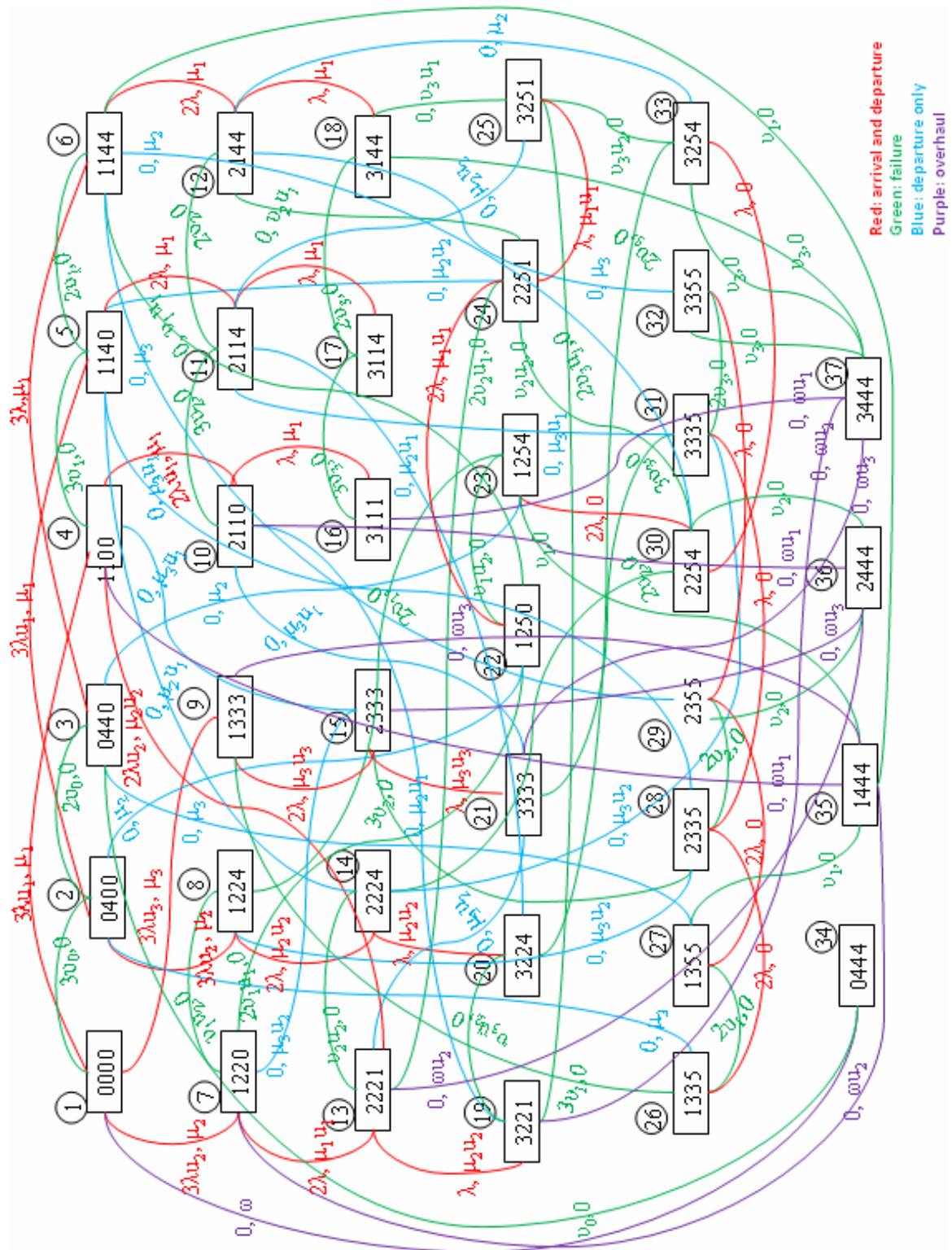


Figure 8.1: Transition rate diagram

9 MATLAB codes and simulations of the airborne Network

9.1 Closed queuing network (reducible)

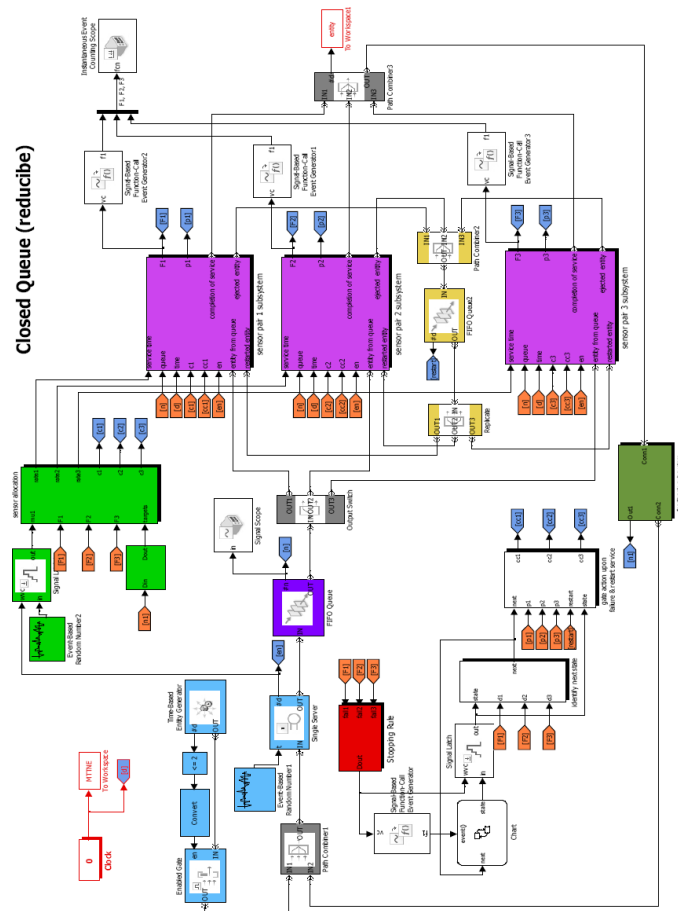


Figure 9.1: Top level Simulink block diagram of simulation model 1

9.1.1 Stateflow diagram of the system state (reducible)

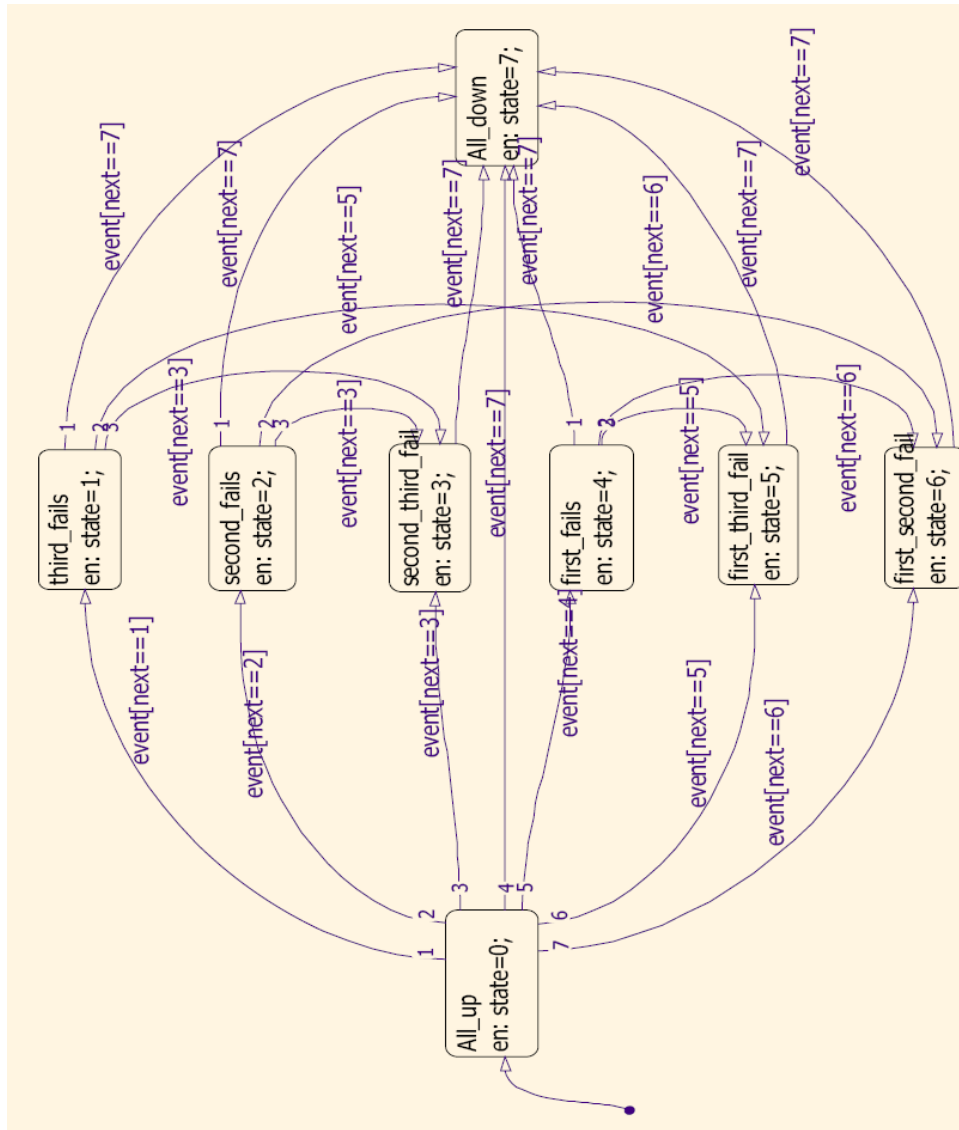


Figure 9.2: System Stateflow diagram of simulation model 1

9.1.2 Stopping rule

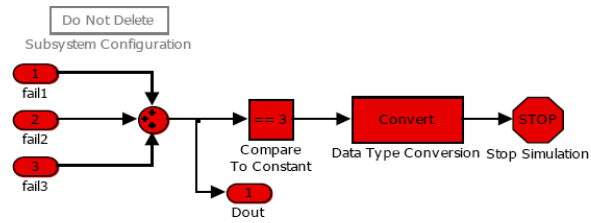


Figure 9.3: Stopping rule of simulation model 1

9.1.3 Computing total number of targets in the system

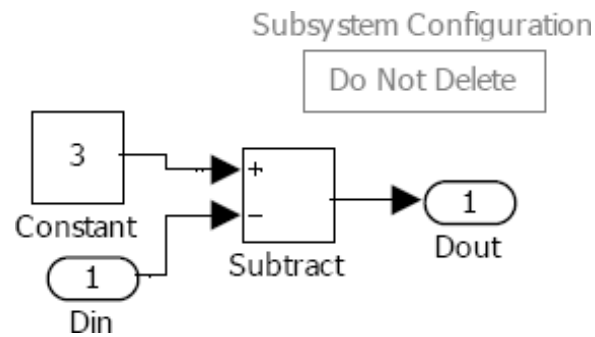


Figure 9.4: Computing total number of targets

9.1.4 Feedback subsystem

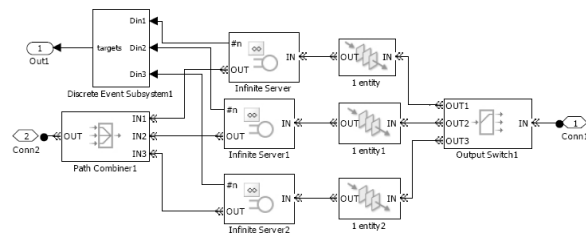


Figure 9.5: Lower portion subsystem

9.1.5 Identify system's next state

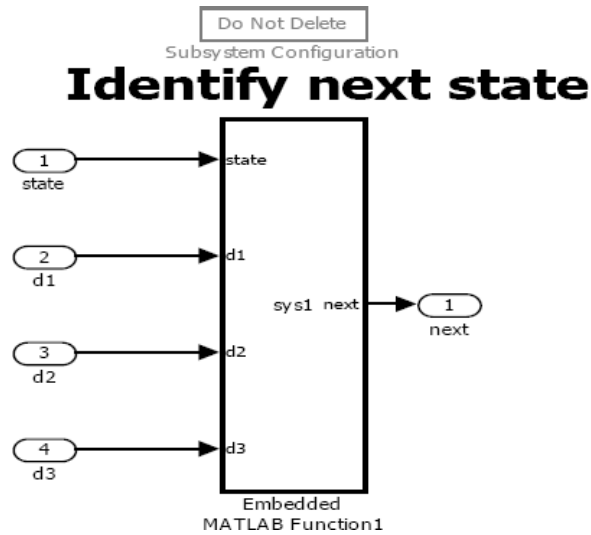


Figure 9.6: Identify next state

The following is the embedded MATLAB code for the next state subsystem:

```
1 function next = sys1(state,d1,d2,d3)
2 persistent last_state;
3
4 % states:(d1,d2,d3), 0 represents sensor up, 1 represents sensor down.
5 % 0 -- 000
6 % 1 -- 001
7 % 2 -- 010
8 % 3 -- 011
9 % 4 -- 100
10 % 5 -- 101
11 % 6 -- 110
12 % 7 -- 111
13
14 % initialize output
15 next=0;
16 switch state
17     case 0
18         xs=[d1;d2;d3];
19         states=[4 2 1];
20         if xs(1)==1 && xs(2)==1 && xs(3)==1
21             next=7;
22         elseif xs(1)==1 && xs(2)==1 && xs(3)==0
23             next=6;
24         elseif xs(1)==1 && xs(2)==0 && xs(3)==1
25             next=5;
```

```

26     elseif xs(1)==0 && xs(2)==1 && xs(3)==1
27         next=3;
28     elseif xs(1)==0 && xs(2)==0 && xs(3)==0
29         next=0;
30     else [x ind]=max(xs);
31         next=states(ind);
32     end
33
34 case 1
35     xs=[d2; d1; -inf];
36     states=[3 5 0];
37     if xs(1)==0 && xs(2)==0
38         next=1;
39     elseif xs(1)==1 && xs(2)==1
40         next=7;
41     else [x ind]=max(xs);
42         next=states(ind);
43     end
44
45 case 2
46     xs=[d3; d1; -inf];
47     states=[3 6 0];
48     if xs(1)==0 && xs(2)==0
49         next=2;
50     elseif xs(1)==1 && xs(2)==1
51         next=7;
52     else [x ind]=max(xs);
53         next=states(ind);
54     end
55
56 case 4
57     xs=[d3; d2; -inf];
58     states=[5 6 0];
59     if xs(1)==0 && xs(2)==0
60         next=4;
61     elseif xs(1)==1 && xs(2)==1
62         next=7;
63     else [x ind]=max(xs);
64         next=states(ind);
65     end
66
67 case 3
68     if d1==1
69         next=7; end
70 case 5
71     if d2==1
72         next=7; end
73 case 6
74     if d3==1
75         next=7; end
76 case 7

```

```

77         next=7;
78     end

```

9.1.6 Service rate generator

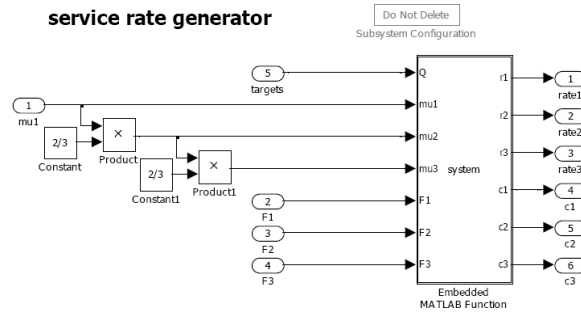


Figure 9.7: Service rate generator

```

1 function [r1,r2,r3,c1,c2,c3] = system(Q,mu1,mu2,mu3,F1,F2,F3)
2 %   Q = queue length
3 %   F1, F2, F3: state of sensors. 0-->UP, 1-->DOWN
4 %   c1,c2,c3: enable gate signal
5 %   This script contains four control policies. Run the simulation
6 %   at each control policy and plot the results together.
7 %-----
8
9 %%GREEDY
10 %   if F1==0 && F2==0 && F3==0
11 %       r1 = mu3; r2=mu3; r3=mu3;
12 %       c1=1; c2=0; c3=0;
13 %   elseif F1==0 && F2==0 && F3==1
14 %       r1 = mu2; r2=mu2; r3=0;
15 %       c1=1; c2=0; c3=0;
16 %   elseif F1==0 && F2==1 && F3==0
17 %       r1 = mu2; r2=0; r3=mu2;
18 %       c1=1; c3=0; c2=0;
19 %   elseif F1==1 && F2==0 && F3==0
20 %       r2 =mu2; r1=0; r3=mu2;
21 %       c2=1; c3=0; c1=0;
22 %   elseif F1==0 && F2==1 && F3==1
23 %       r1 = mu1; r2=0; r3=0;
24 %       c1=1; c2=0; c3=0;
25 %   elseif F1==1 && F2==0 && F3==1
26 %       r2 = mu1; r1=0; r3=0;
27 %       c2=1; c1=0; c3=0;
28 %   elseif F1==1 && F2==1 && F3==0
29 %       r3 = mu1; r1=0; r2=0;

```

```

30 %           c3=1; c1=0; c2=0;
31 %     else r1=0; r2=0; r3=0;
32 %           c1=0; c2=0; c3=0;
33 %     end
34 %%-----
35
36 % %% CONSERVATIVE
37 %   if F1==0 && F2==0 && F3==0
38 %       r1 = mul; r2 = mul; r3 =mul;
39 %       c1=1; c2=1; c3=1;
40 %   elseif F1==0 && F2==0 && F3==1
41 %       r1 = mul; r2= mul; r3 = mul;
42 %       c1=1; c2=1; c3=0;
43 %   elseif F1==0 && F2==1 && F3==0
44 %       r1 = mul; r3 = mul; r2 =mul;
45 %       c1=1; c3=1; c2=0;
46 %   elseif F1==1 && F2==0 && F3==0
47 %       r2 =mul; r3 = mul; r1 = mul;
48 %       c2=1; c3=1; c1=0;
49 %   elseif F1==0 && F2==1 && F3==1
50 %       r1 = mul; r2 = mul; r3 = mul;
51 %       c1=1; c2=0; c3=0;
52 %   elseif F1==1 && F2==0 && F3==1
53 %       r2 = mul; r1 = mul; r3 = mul;
54 %       c2=1; c1=0; c3=0;
55 %   elseif F1==1 && F2==1 && F3==0
56 %       r3 = mul; r1 = mul; r2 = mul;
57 %       c3=1; c1=0; c2=0;
58 %   else r1=0; r2=0; r3=0;
59 %       c1=0; c2=0; c3=0;
60 %   end
61 %%-----
62
63 % AGGRESSIVE
64 %   if F1==0 && F2==0 && F3==0 && Q==0
65 %       r1 = mu3; r2=mu3; r3=mu3;
66 %       c1=1; c2=1; c3=1;
67 %   elseif F1==0 && F2==0 && F3==0 && Q==1
68 %       r1 = mu3; r2=mu3; r3=mu3;
69 %       c1=1; c2=0; c3=0;
70 %   elseif F1==0 && F2==0 && F3==0 && Q==2
71 %       r1 = mu2; r2=mu2; r3=mul;
72 %       c1=1; c2=0; c3=1;
73 %   elseif F1==0 && F2==0 && F3==0 && Q>=3
74 %       r1 = mul; r2=mul; r3=mul;
75 %       c1=1; c2=1; c3=1;
76 %
77 %   elseif F1==0 && F2==0 && F3==1 && Q==0
78 %       r1 = mu2; r2=mu2; r3=0;
79 %       c1=1; c2=1; c3=0;
80 %   elseif F1==0 && F2==0 && F3==1 && Q==1

```

```

81 %      r1 = mu2; r2=mu2; r3=0;
82 %      c1=1; c2=0; c3=0;
83 % elseif F1==0 && F2==0 && F3==1 && Q==2
84 %      r1 = mul; r2=mul; r3=mul;
85 %      c1=1; c2=1; c3=0;
86 % elseif F1==0 && F2==0 && F3==1 && Q>=3
87 %      r1 = mul; r2=mul; r3=mul;
88 %      c1=1; c2=1; c3=0;
89 %
90 % elseif F1==0 && F2==1 && F3==0 && Q==0
91 %      r1 = mu2; r2=0; r3=mu2;
92 %      c1=1; c3=0; c2=1;
93 % elseif F1==0 && F2==1 && F3==0 && Q==1
94 %      r1 = mu2; r2=0; r3=mu2;
95 %      c1=1; c3=0; c2=0;
96 % elseif F1==0 && F2==1 && F3==0 && Q==2
97 %      r1 = mul; r2=0; r3=mul;
98 %      c1=1; c3=1; c2=0;
99 % elseif F1==0 && F2==1 && F3==0 && Q>=3
100 %      r1 = mul; r2=0; r3=mul;
101 %      c1=1; c3=1; c2=0;
102 %
103 % elseif F1==1 && F2==0 && F3==0 && Q==0
104 %      r2 =mu2; r1=0; r3=mu2;
105 %      c2=1; c3=1; c1=0;
106 % elseif F1==1 && F2==0 && F3==0 && Q==1
107 %      r2 =mu2; r1=0; r3=mu2;
108 %      c2=1; c3=0; c1=0;
109 % elseif F1==1 && F2==0 && F3==0 && Q==2
110 %      r2 =mul; r1=0; r3=mul;
111 %      c2=1; c3=1; c1=0;
112 % elseif F1==1 && F2==0 && F3==0 && Q>=3
113 %      r2 =mul; r1=0; r3=mul;
114 %      c2=1; c3=1; c1=0;
115 %
116 % elseif F1==0 && F2==1 && F3==1
117 %      r1 = mul; r2=0; r3=0;
118 %      c1=1; c2=0; c3=0;
119 %
120 % elseif F1==1 && F2==0 && F3==1
121 %      r2 = mul; r1=0; r3=0;
122 %      c2=1; c1=0; c3=0;
123 %
124 % elseif F1==1 && F2==1 && F3==0
125 %      r3 = mul; r1=0; r2=0;
126 %      c3=1; c1=0; c2=0;
127 % else r1=0; r2=0; r3=0;
128 %      c1=0; c2=0; c3=0;
129 % end
130 %-----
131 %%Optimal 1

```

```

132     if F1==0 && F2==0 && F3==0 && Q==0
133         r1 = mu2; r2=mu2; r3=mu2;
134         c1=1; c2=1; c3=1;
135     elseif F1==0 && F2==0 && F3==0 && Q==1
136         r1 = mu2; r2=mu2; r3=mul;
137         c1=1; c2=0; c3=1;
138     elseif F1==0 && F2==0 && F3==0 && Q==2
139         r1 = mu2; r2=mu2; r3=mul;
140         c1=1; c2=0; c3=1;
141     elseif F1==0 && F2==0 && F3==0 && Q>=3
142         r1 = mu2; r2=mu2; r3=mul;
143         c1=1; c2=0; c3=1;
144
145     elseif F1==0 && F2==0 && F3==1 && Q==0
146         r1 = mu2; r2 =mu2; r3 = mul;
147         c1=1; c2=0; c3=0;
148     elseif F1==0 && F2==0 && F3==1 && Q==1
149         r1 = mu2; r2 =mu2; r3 = mul;
150         c1=1; c2=0; c3=0;
151     elseif F1==0 && F2==0 && F3==1 && Q>=2
152         r1 = mul; r2 =mul; r3 = mul;
153         c1=1; c2=1; c3=0;
154
155     elseif F1==0 && F2==1 && F3==0 && Q==0
156         r1 = mu2; r2=0; r3=mu2;
157         c1=1; c3=0; c2=0;
158     elseif F1==0 && F2==1 && F3==0 && Q==1
159         r1 = mu2; r2=0; r3=mu2;
160         c1=1; c3=0; c2=0;
161     elseif F1==0 && F2==1 && F3==0 && Q>=2
162         r1 = mul; r2=0; r3=mul;
163         c1=1; c3=1; c2=0;
164
165     elseif F1==1 && F2==0 && F3==0 && Q==0
166         r2 =mu2; r1=0; r3=mu2;
167         c2=1; c3=0; c1=0;
168     elseif F1==1 && F2==0 && F3==0 && Q==1
169         r2 =mu2; r1=0; r3=mu2;
170         c2=1; c3=0; c1=0;
171     elseif F1==1 && F2==0 && F3==0 && Q>=2
172         r2 =mul; r1=0; r3=mul;
173         c2=1; c3=1; c1=0;
174     elseif F1==0 && F2==1 && F3==1
175         r1 = mul; r2=0; r3=0;
176         c1=1; c2=0; c3=0;
177     elseif F1==1 && F2==0 && F3==1
178         r2 = mul; r1=0; r3=0;
179         c2=1; c1=0; c3=0;
180     elseif F1==1 && F2==1 && F3==0
181         r3 = mul; r1=0; r2=0;
182         c3=1; c1=0; c2=0;

```

```

183     else r1=0; r2=0; r3=0;
184         c1=0; c2=0; c3=0;
185     end

```

9.1.7 New allocation for return targets

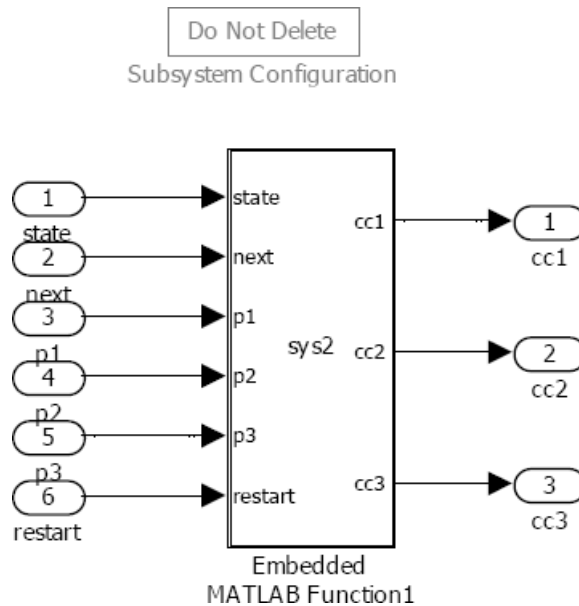


Figure 9.8: New allocation for return targets

```

1 function [cc1,cc2,cc3] = sys2(state,next,p1,p2,p3,restart)
2
3 % p1,p2,p3 are the ejected entities from servers
4 % Q: upper system queue length, include the ones in service
5 % state: current state of the system
6 % next: next state of the system
7 % cc1,cc2,cc3: signals determine which server serves a restarted entity
8 % -----
9
10 if restart >=1
11     if state ==0 && next==1 && p3==1
12         cc1=1; cc2=0; cc3=0;
13     elseif state==0 && next==2 && p2==1
14         cc1=1; cc2=0; cc3=0;
15     elseif state==0 && next==4 && p1==1
16         cc1=0; cc2=1; cc3=0;
17
18     elseif state ==1 && next==3 && p2==1
19         cc1=1; cc2=0; cc3=0;

```

```

20     elseif state ==1 && next==5 && p1==1
21         cc1=0; cc2=1; cc3=0;
22
23     elseif state==2 && next==3 && p3==1
24         cc1=1; cc2=0; cc3=0;
25     elseif state==2 && next==6 && p1==1
26         cc1=0; cc2=0; cc3=1;
27
28     elseif state==4 && next==5 && p3==1
29         cc1=0; cc2=1; cc3=0;
30     elseif state==4 && next==6 && p2==1
31         cc1=0; cc2=0; cc3=1;
32
33     elseif state==3 && next==7
34         cc1=0; cc2=0; cc3=0;
35
36     elseif state==5 && next==7
37         cc1=0; cc2=0; cc3=0;
38
39     elseif state==6 && next==7
40         cc1=0; cc2=0; cc3=0;
41
42     else cc1=0; cc2=0; cc3=0;
43     end
44 else cc1=0; cc2=0; cc3=0;
45 end

```

9.1.8 Sensor-pair subsystem

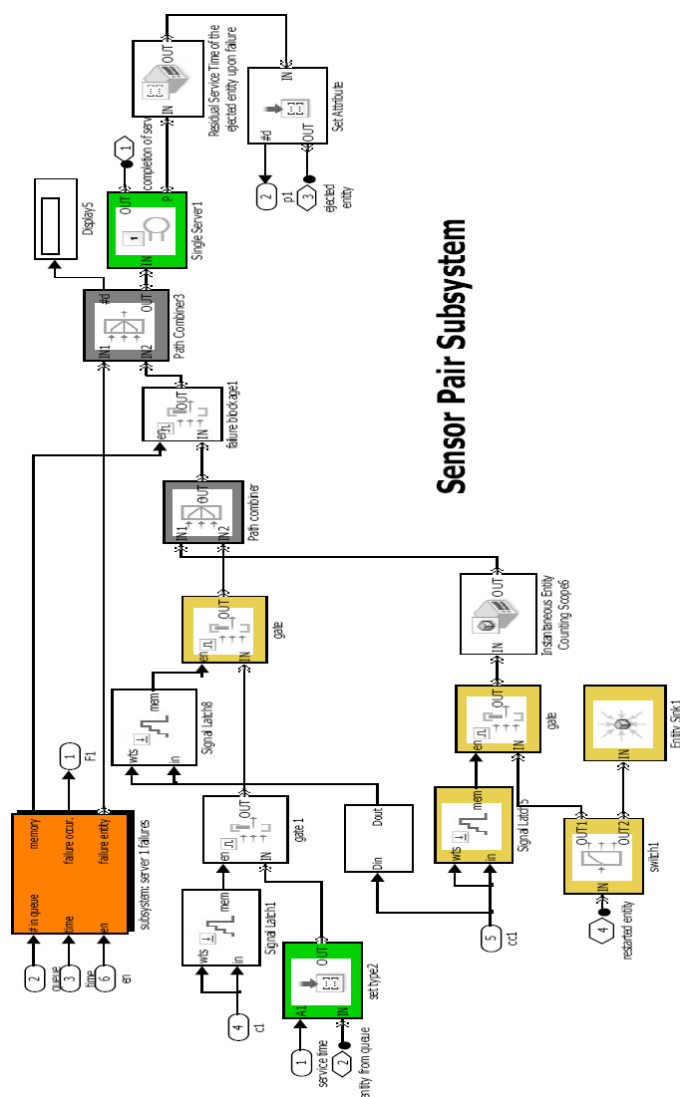
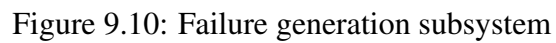


Figure 9.9: Sensor-pair subsystem for simulation model 1

The system has three identity sensor-pairs subsystem.

All sensor-pairs have same random failure time; therefore all sensor-pairs subsystems contain the following subsystem.



9.1.9.1 Discrete event subsystem1

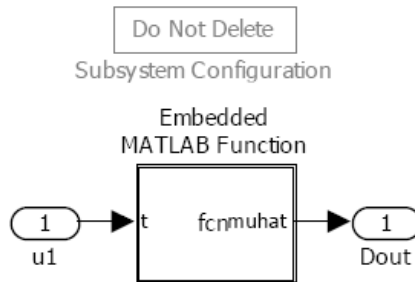


Figure 9.11: Discrete event subsystem1

This block is within the failure generation subsystem, follow code is embedded in the block. The code generates exponential failure time of the sensor-pairs.

```
1 function muhat = fcn(t)
2 persistent oldtime;
3 %initialize output
4 muhat=2000;
5 %initialize persistent variable
6 if isempty(oldtime)
7     oldtime=0;
8 end
9 %update
10 if (t>oldtime)
11     oldtime=t;
12     muhat= expfit(exprnd(2000,10,1));
13 end
```

9.1.9.2 Discrete event subsystem 2

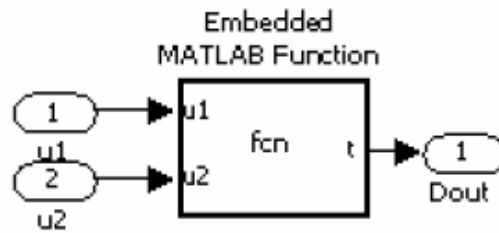


Figure 9.12: Discrete event subsystem 2

The above block is also within failure generation subsystem, following code is embedded in the block which assigns failure time to failure entity.

```
1 function t = fcn(u1,u2)
2 persistent oldtime;
3 %initialize output
4 t=0;
5 %initialize persistent variable
6 if isempty(oldtime)
7     oldtime=0;
8 end
9 %update
10 if (u1>oldtime)
11     oldtime=u1;
12 end
13 if u1≥u2
14     t=0;
15 else t=u2-oldtime;
16 end
```

9.1.9.3 Discrete event subsystem 3

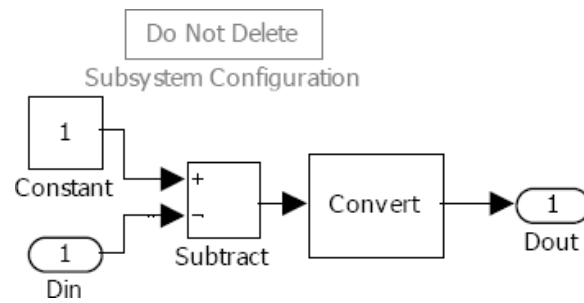


Figure 9.13: Discrete event subsystem 3

If there is an entity exits the server in the failure generation subsystem of a sensor-pair, then that sensor-pair fails. Otherwise, the sensor-pair is still up.

9.2 Irreducible case with system replenishment and with or without channel fading

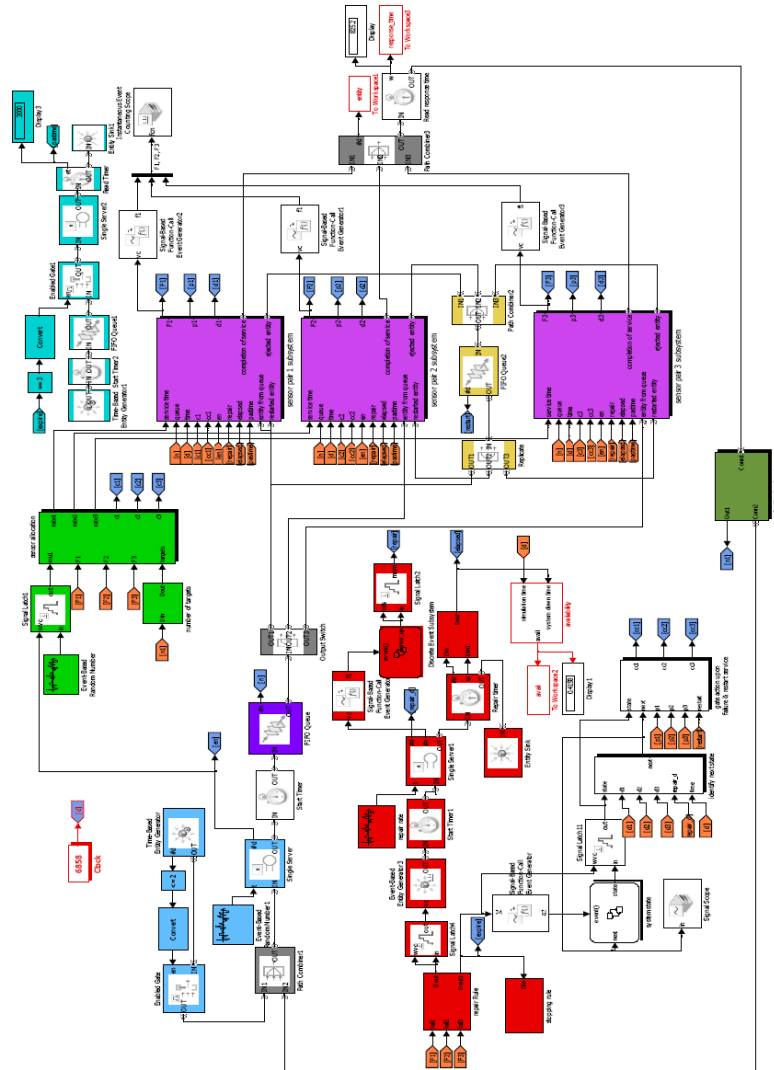


Figure 9.14: Top level Simulink block diagram for simulation model 2 and 3.

9.2.1 Stateflow diagram of the system (irreducible)

The stateflow diagram is similar to fig C.2, except it has a transition from final failure state to initial state.

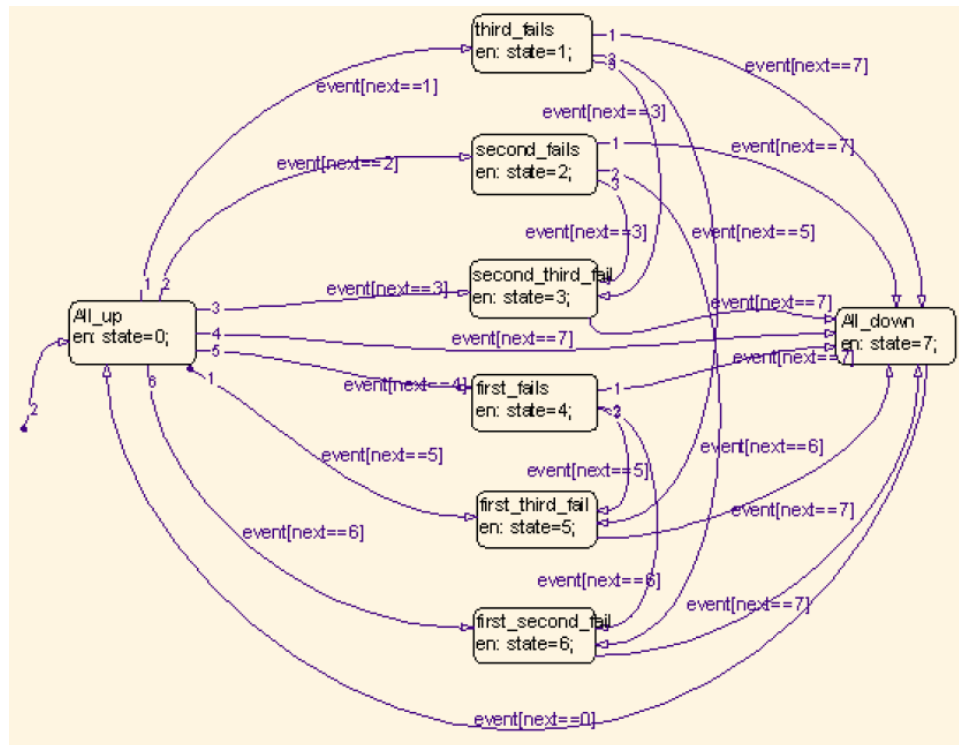


Figure 9.15: System state-flow diagram for simulation model 2 and 3.

9.2.2 System up/down stateflow diagram

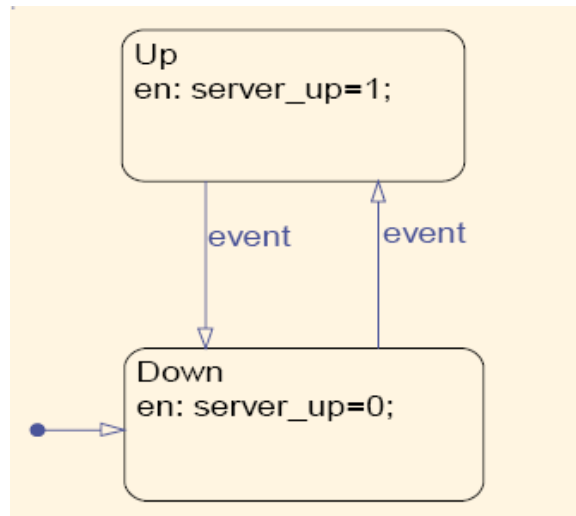


Figure 9.16: State-flow diagram of network status.

9.2.3 Stopping rule (irreducible)

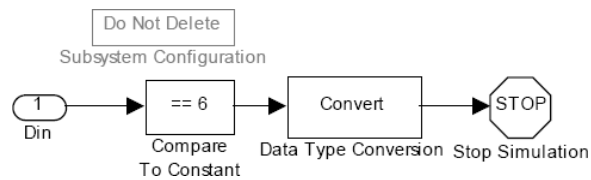


Figure 9.17: Stopping rule for simulation model 2 and 3.

9.2.4 Repairing rule

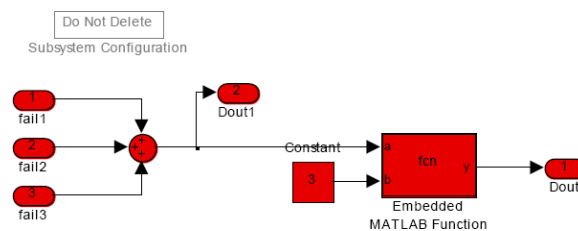


Figure 9.18: Repairing rule of simulation model 2 and 3.

9.2.5 Sensor-pair subsystem (irreducible) without channel fading

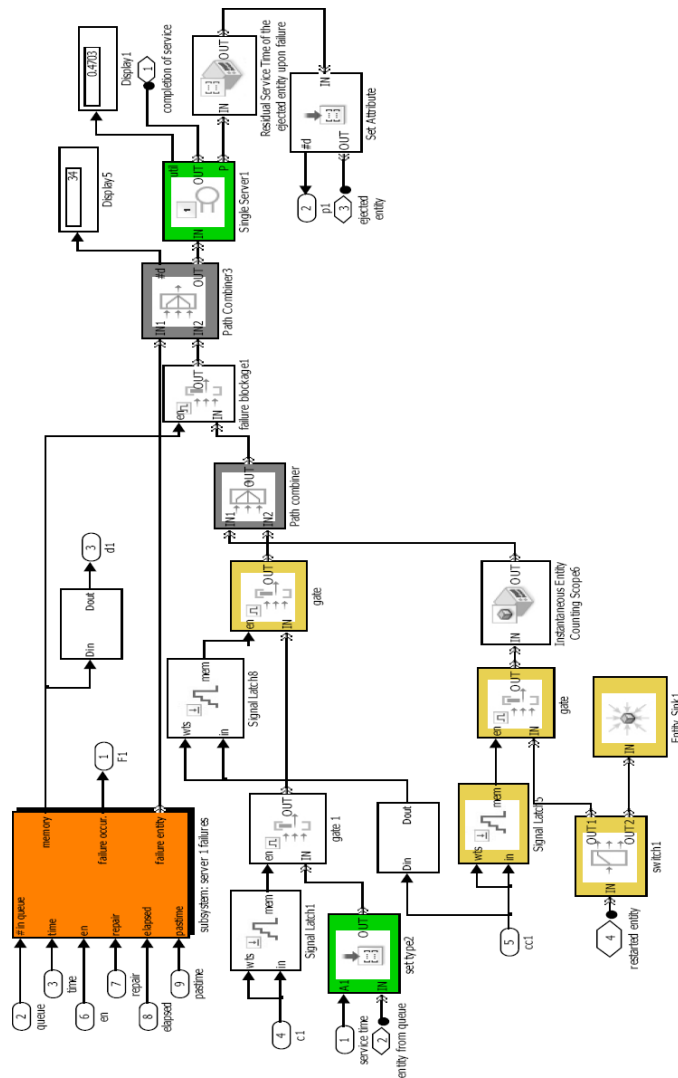


Figure 9.19: Sensor-pair subsystem without channel fading

9.2.6 Failure generation subsystem (irreducible)

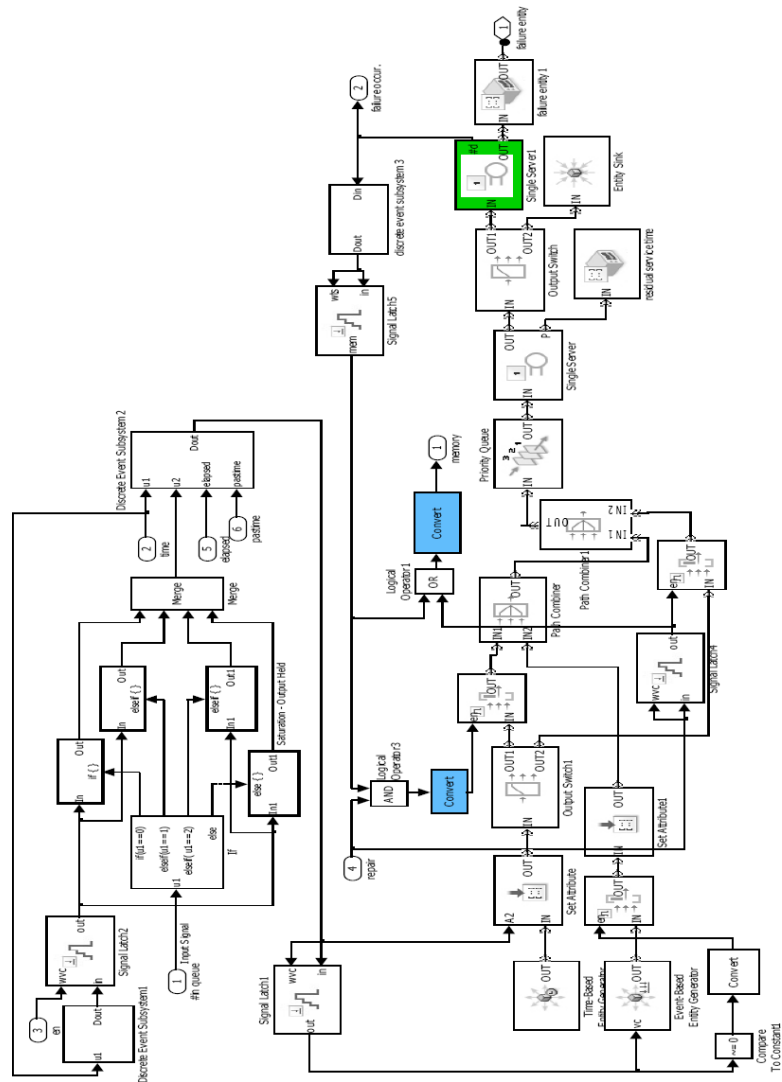


Figure 9.20: Failure generation subsystem for simulation model 2 and 3.

9.2.6.1 Discrete event subsystem 2

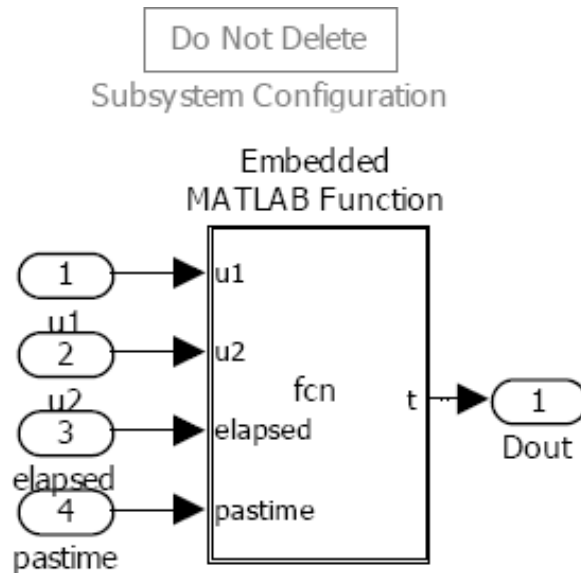


Figure 9.21: This block is within Fig. C.16, the function of this block generates failure time.

```

1 function t = fcn(u1,u2,elapsed,pastime)
2 persistent oldtime;
3 %initialize output
4 t=0;
5 %initialize persistent variable
6 if isempty(oldtime)
7     oldtime=0;
8 end
9 %update
10 if (u1>oldtime)
11     oldtime=u1;
12 end
13 T = elapsed+pastime;
14 if u2 ≤ u1-T
15     t=0;
16 else t=u2-u1+T;
17 end

```

9.2.6.2 Discrete event subsystem 3

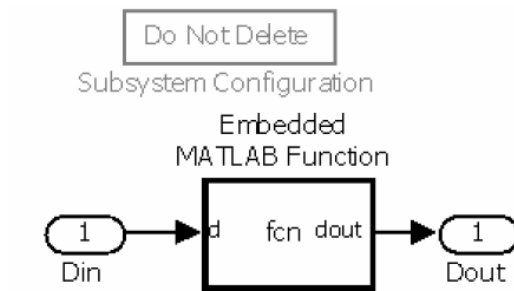


Figure 9.22: Discrete event subsystem 3

```
1 function dout=fcn(d)
2
3 if d==0
4     dout=1;
5 else dout=0;
6 end
```

9.3 Irreducible case with system replenishment and intermittent channel fading

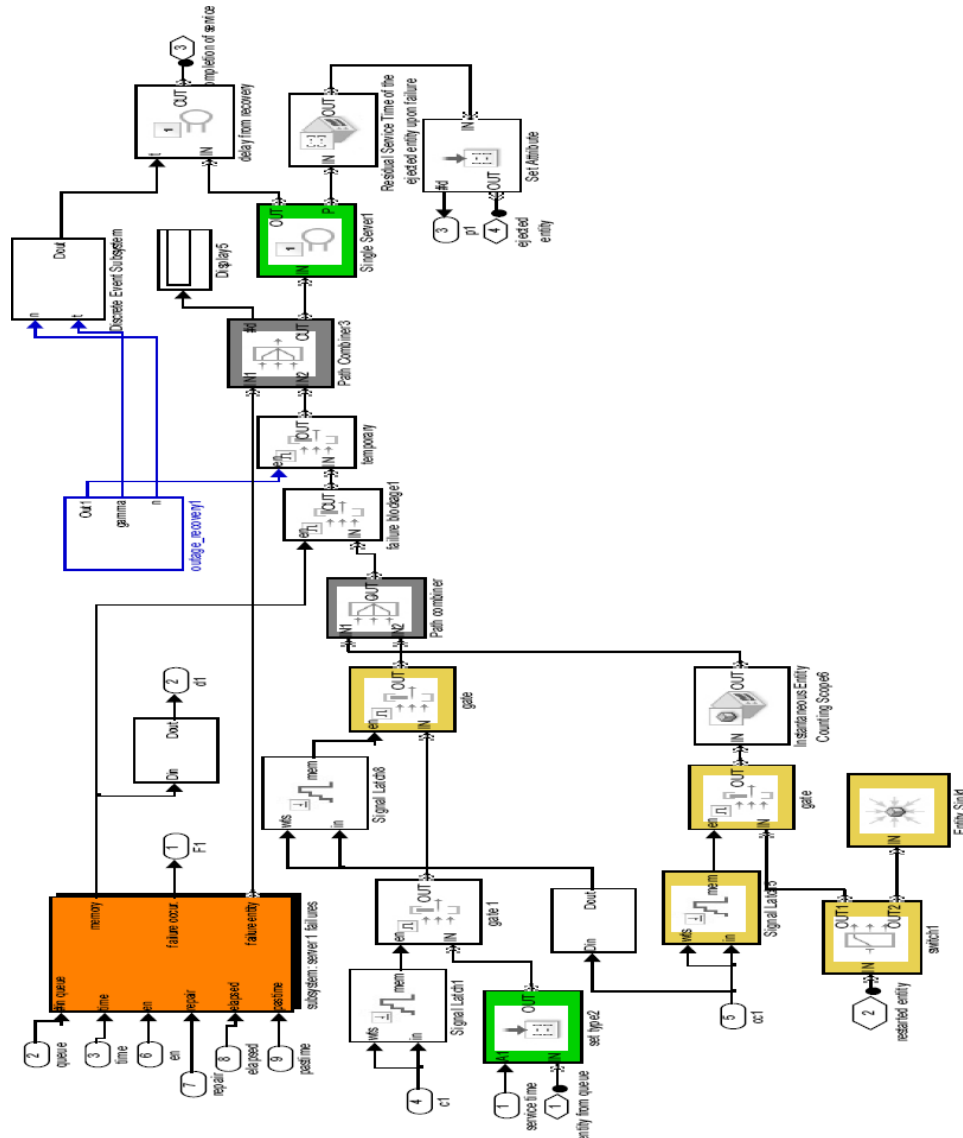


Figure 9.23: Sensor-pair subsystem of simulation model 3

Figure C.18 shows how to setup the temporary outage and how to recovery the loss data for the sensor pairs and figure C.19 determines the recovery rate.

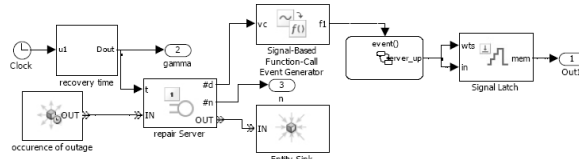


Figure 9.24: Outage and recovery subsystem

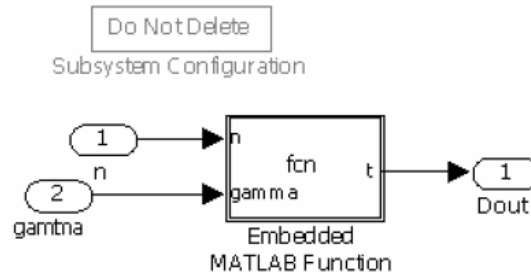


Figure 9.25: Discrete-event subsystem for recovery rate

```

1 function t = fcn(n, gamma)
2 if n==0
3     t=0;
4 else t=gamma;
5 end

```

9.4 Code for running the simulations repeatedly

9.4.1 Initial seed value generator:

```

1 % Generate a vector of large odd numbers.
2 newseed = (30000 : 2 : 79999);
3
4 % Randomly permute the numbers to avoid always using the same set of seeds.
5 perm = randperm(length(newseed));
6 paramname = {'initialseed', 'seed'}; % Parameter names to consider
7 np = length(paramname);
8 idx = 1;
9
10 for jj=1:np
11
12     % Find blocks that have the parameter with a numerical value
13     blocks = find_system(bdroot, 'RegExp', 'on', 'LookUnderMasks', 'all', ...
14         paramname{jj}, '\d+');
15

```

```

16     % Replace initial seed parameter values with numbers from the vector.
17     for kk = 1:length(blocks)
18         newseedparamvalue = num2str(newseed(perm(idx)));
19         idx = idx + 1;
20         set_param(blocks{kk},paramname{jj},newseedparamvalue);
21         disp(['Setting parameter to ' newseedparamvalue ' in ' ...
22             strrep(blocks{kk},char(10),' ') '.']]);
23     end
24 end

```

9.4.2 Run simulation repeated to gather results and varying a parameter

9.4.2.1 1. Mean time to network expiration

```

1  % execute the following two comments in MATLAB, resave the model, then
2  % start the simulation.
3  % set_param('closetest1','PostLoadFcn','')
4  % set_param('closetest1','CloseFcn','')
5
6  clear global all;
7
8  load_system('closetest1'); % Load system
9  nruns = 30; % Number of simulation runs
10
11 %initialization
12 w = zeros(nruns,1); %MTTNE
13 d = zeros(nruns,1); %# of entity departed
14 opts = simset('SrcWorkspace','Current','DstWorkspace','Current');
15 h = waitbar(0,'Running simulations. Please wait...Be patient');
16 seedarray1 = ceil(rand(nruns,1)*99999)*2+1;
17 seedarray2 = ceil(rand(nruns,1)*99999)*2+1;
18
19 % Vary the mean arrival rate.
20 la = linspace(1/300,0.02,4); % Values of mean arrival rate
21 wavg = zeros(length(la),1);
22 davg = zeros(length(la),1);
23 for i = 1:length(la)
24     % m is a parameter in the random number block, so changing m changes the
25     % mean arrival rate in the simulation.
26     m = la(i);
27     disp(['Simulating with mean arrival rate=' num2str(m)]);
28
29 % Replicate for each value of m
30     for k = 1:nruns
31         waitbar(k/nruns,h);
32         seedvalue1 = seedarray1(k);
33         seedvalue2 = seedarray2(k);
34         sim('closetest1',[],opts); % Run simulation.
35         w(k) = MTTNE;

```

```

36         d(k) = entity;
37     end
38     wavg(i) = mean(w);           % Average for fixed value of m
39     davg(i) = mean(d);
40 end
41 close(h);
42 disp('average # of entity departed: ')
43 disp(davg)
44 disp(wavg)
45 %% plot avg. MTTNE
46 figure;
47 plot(la,wavg,'-o');
48 title('Mean Time to Network Failure vs. Arrival Rate')
49 xlabel('Arrival Rate'), ylabel('MTTNE (sec)')

```

9.4.2.2 Response time and availability

The following code is for design 2, may change the model name for design 3.

```

1  % execute the following two comments in MATLAB, resave the model, then
2  % start the simulation.
3  %set_param('closed_replenishment3','PostLoadFcn','')
4  %set_param('closed_replenishment3','CloseFcn','')
5
6  clear global all;
7
8  load_system('closed_replenishment3'); % Load system
9  nruns = 30; % Number of simulation run
10
11 %initialization
12 R = zeros(nruns,1); %response time
13 A = zeros(nruns,1); %availability
14 opts = simset('SrcWorkspace','Current','DstWorkspace','Current');
15 h = waitbar(0,'Running simulations. Please wait...');
16 seedarray1 = ceil(rand(nruns,1)*99999)*2+1;
17 seedarray2 = ceil(rand(nruns,1)*99999)*2+1;
18 seedarray3 = ceil(rand(nruns,1)*99999)*2+1;
19
20 la = linspace(1/300,1/50,4); % Values of mean arrival rate
21 Ravg = zeros(length(la),1);
22 Aavg = zeros(length(la),1);
23
24 % Vary the mean arrival rate.
25 for i = 1:length(la)
26     % m is a parameter in the random number block, so changing m
27     % changes the mean arrival rate in the simulation.
28     m = la(i);
29     disp(['Simulating with mean arrival rate=' num2str(m)]);
30
31     % Replicate for each value of m

```

```

32     for k = 1:nruns
33         waitbar(k/nruns,h);
34         seedvalue1 = seedarray1(k);
35         seedvalue2 = seedarray2(k);
36         seedvalue3 = seedarray3(k);
37         sim('closed_replenishment3',[],opts); % Run simulation.
38         R(k) = response_time;
39         A(k) = avail;
40     end
41     Ravg(i) = mean(R); % Average for fixed value of R and A
42     Aavg(i) = mean(A);
43 end
44 close(h)
45
46 % plot response time and availability
47 figure(1), plot(la,Ravg,'-o');
48 title('Response Time vs. Arrival Rate')
49 xlabel('Arrival Rate'), ylabel('Response Time (sec)')
50 figure(2), plot(la,Aavg,'-x');
51 title('Availability vs. Arrival Rate')
52 xlabel('Arrival Rate'), ylabel('Availability')

```

10 Symbols, Abbreviations and Acronyms

FDOA Frequency Difference of Arrival

MTTNE Mean Time to Network Expiration

TDOA Time Difference of Arrival

UAV Unmanned Aerial Vehicle