A COMPARATIVE STUDY OF SOLAR INTERFERENCE ON THE IRIDIUM AND MSS CONSTELLATIONS

Technical Report 93-11

Tsangmin Chang

Department of Computer Science and Engineering
Auburn University
Auburn University, AL 36849-5347

September 7, 1993

A Technical Report
on

A Comparative Study Of Solar Interference On The Iridium And MSS Constellations

May 21, 1993

Tsangmin Chang

Department of

Computer Science and Engineering

Auburn University

Auburn, AL 36849

ABSTRACT

Radio frequency cross-links are the preferred communication medium for low earth orbit (LEO) satellite networks today. However, future cross-link designs may be expected to deploy laser cross-links to increase communication bandwidth. These cross-links are rendered inoperative when incident with high level background radiation sources such as the Sun. The Iridium and Multiple Satellite System (MSS) are two well researched constellations in the literature and are ideal candidates for the investigation of solar interference on point-to-point constellations. In both cases, the impact of solar radiation on link availability is the goal of this project.

Evaluation of link availability based on the motions of the bodies in question is computationally infeasible. It is therefore necessary to reduce the state space to a manageable computing basis by reformulating the problem domain and by simulation. In this report the development of an algorithm and a software simulation tool to analyze the two constellations will be presented and the results of laser cross-link availability will be discussed. The technique introduced here may be generalized to study a variety of different constellations.

Table of Contents

1 INTRODUCTION

Using laser cross-links in low earth orbit (LEO) satellite networks permits high communication bandwidths, is more secure (reduces the opportunity to intercept data) and requires less power than their microwave counterparts. Their use also introduces many new and interesting research problems not addressed in conventional methods. One such problem is communication disruption caused by external noise sources (e.g., background radiation from the Sun and Moon) on the laser-cross links.

Each laser cross-link communication channel is maintained by a transmitter (laser or laser diode) and receiver (photodetector) pair. Whenever the photodetectors are incident with a concentrated light source, such as the Sun or Moon, they must be protected, otherwise physical damage to the detectors will occur. The light sources will be generically referred to as fixed position noise sources and clearly play an important role in communication performance as the constellation changes with time. During the period that the sensor is incident to a noise source, the link using this sensor will be broken and will remain in this state until it is safe to re-establish communications. This results in increased traffic elsewhere in the network. It is assumed that light reflections off the surface and atmosphere of the Earth will not effect the photodiode sensors since the satellite receivers physically prevent the sensor from being incident with the Earth.

In section 2 an introduction to the Iridium and MSS constellations will be given. In section 3 the parameters of interest, metric of measurement and reduction of simulation space will be studied. In section 4 the basic mathematical formulae, the simulation tool used and the algorithm used to implement the study will be explained. In section 5 the Iridium and MSS simulation results will be presented and compared with one another. Section 6 concludes the paper and provides the highlights of the study.

2 CONSTELLATION CHARACTERS

There have been a variety of LEO systems envisaged in the literature. These include the Global Positioning System (GPS)[5, 7]; Motorola's project Iridium[3, 8], designed to provide global telecommunications;

the European LEONET[9], a network of satellites providing global coverage and the Defense Department's Multiple Satellite System (MSS)[4, 5, 6, 8].

In this paper the Iridium project constellation and MSS are selected for analysis. The Iridium constellation was chosen because: i) the constellation has a high degree of regularity which permits a reduction of the state space based on symmetry; ii) the current impetus in high capacity communications makes Iridium a viable model; iii) many of the specifications are available in the public arena. MSS was selected because of its unique property of offering a "random" constellation. This provides an ideal reference constellation with which more regular constellations may be compared.

Model Assumptions

The satellites are assumed to be in an idealized circular orbit and move independently of one another as defined by their orbital elements. The constellation specifications dictate that the satellites are evenly distributed to cover the globe. The only link outages are those caused by background radiation from the Sun and Moon. It is these outages that we consider when evaluating satellite constellations.

The Iridium Constellation

"Iridium" is the name given by the Motorola Corporation to its proposed worldwide satellite-based cellular phone system[3, 8]. This system will provide the user with the ability to make and to receive calls from any point on the Earth.

The constellation chosen by a Motorola design team [1] employs the satellites in circular polar orbits, with 11 satellites in each of 7 orbital planes. The constellation is co-rotating (all satellites move towards the North pole on one side of the Earth and towards the South pole on the other) with adjacent planes out of phase with one another (i.e. planes 1,3,5,7 are in phase, planes 2,4,6 are in phase, planes 1,2 are out of phase). Every satellite has four links which are linked with upper, low, left, and right neighbor satellites. There are no links between plane 1 and plane 7. The constellation and link assignments are portrayed in Figure 1. A viewing horizon of 10° or more is required for contact between the satellite antenna and a portable unit, because of the fixed-geometry beams employed by the satellites.

The satellites are at an altitude of 413.5 n. mi. Each of the seven co-rotating planes is separated by slightly more than 27°. The seam between planes 1 and 7, which represents satellites going in opposite directions, is separated by approximately 17°.

The constellation cross-links will be arranged in a cubic mesh, with each satellite connected to the satellites immediately behind and in front in the same plane. Up to four inter plane cross-links must be maintained with a maximum distance of 2000 n. mi at all times. This is accomplished by connecting the satellites closest to the poles in a regular fashion to satisfy the above constraints.

The switched digital communications system is designed for first operation in 1996. Currently the system is designed to use four 50MHz L-Band cross-links (22.55 to 23.55 GHz) and 100MHz K band down/up links (18.8-20.2 GHz and 27.5-30.0 GHz).

The Iridium constellation was modelled using an orbit generation function (Mil-3's genorb utility) based on the Artificial Satellite Analysis Program (ASAP), version 2.0, written by J. H. Kwok at the Jet Propulsion Laboratory in Pasadena, California. The model includes oblateness of the Earth, effects of the Sun and Moon, atmospheric drag, solar radiation pressure, and satellite mass.

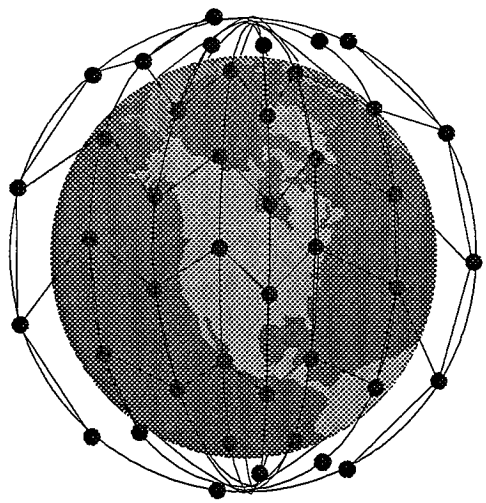

Figure 1 The Iridium Constellation

The MSS Constellation

The Multiple Satellite System (MSS) [4, 5, 6, 8] is designed to be a highly survivable communication system tailored to defense applications. The constellation considered consists of a large number of satellites (240) in approximately random orbits relative to one another, at altitudes of 350 to 400 n.mi. The constellation is defined by three sets of 80 satellites launched at deployment inclination angles of 27.5°, 57.5°, and 90°. Because of the different altitudes, the satellites will tend to become more or less uniformly distributed over the globe. The constellation is therefore "random" to all practical purposes.

The cross link subsystem is characterized by a topology in which each satellite typically may communicate with several neighboring satellites. The actual number varies with time and depends on the satellites neighbors. The maximum cross-link distance is limited to 1200 n.mi. and each satellite may be connected to at most six neighbors. The actual link assignment algorithm is not available in public literature but is assumed to be a random connection to neighboring satellites. Average cross link lifetime under these conditions is typically 7 minutes.

The end-to-end delay through the system for short packets will be less than 200 msec to satisfy voice and interactive data requirements. The system will also degrade gracefully, with reduced capacity communications continuing to be provided to all geographic areas of interest for at least a 50–percent satellite depletion. During peacetime the system will be used as a general-purpose digital network providing global voice-data communications. Depending on the traffic matrix, some satellites may be handling only cross link traffic while others may be handling only up/down link traffic. Typically satellites will be handling both. Uplink and downlink data rates of 1Mbps are envisaged.

The constellation was modelled using the same orbit generating utility as Iridium.

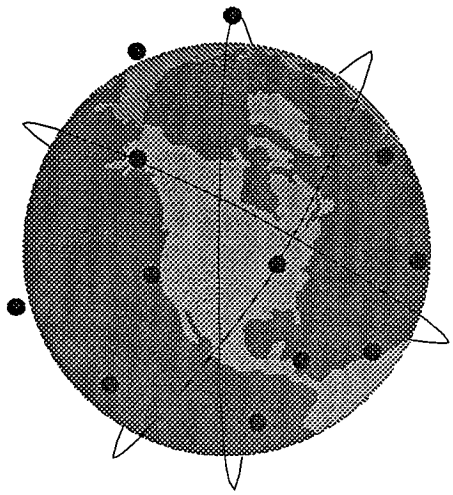

Figure 2 Constellation and Link Assignment for MSS

3 PERFORMANCE ANALYSIS

Parameter Considerations

The parameters of interest are temporal/spatial coordinates, field of view (FOV), and background radiation noise source positions and durations. The temporal/spatial coordinates are functions of the constellation under consideration. The duration of cross-link activity in a noiseless environment is dependent upon the constellation and the FOV of the transmitter/receiver pairs. When the background noise is interjected into the analysis, the efficiency of the network is reduced due to cross-links being blocked and the resulting increase in data flow through alternate paths in the network.

Before proceeding it is appropriate to consider the selection of time versus orbital position as the independent variable in the analysis. It is desirable to minimize the sample space of analysis and still maintain a high confidence level in the data. If time were used and the periods of the satellite constellation, Moon, and Earth orbit were taken into account, a simulation time of at least 365 days would be required in order to cover the entire possible spectrum of data. It is easy to see that this would generate large data sets, not easily manageable by the available computer hardware. If we limited the time period, then we have to ensure that the time period enabled all possible variations of input data.

Alternately, satellite positions in a spherical coordinate system could be used as the independent variable. This approach seems more plausible since it is the relative positions of the extraneous noise inputs that cause link/satellite failures. These positions can be thought of as coordinate tuples. Considering the relative distances, the unit of measure can readily be limited to one degree or less. If we assume that the solar (or lunar) rays arrive colinear with the line joining the satellite and the Sun (or Moon), then the distances of the Sun (Moon) from the Earth and satellites are considered to be infinite.

As the satellites become colinear with the rays of the background radiation sources, the range of positions for which the satellite link is disabled becomes an important factor in determining the effectiveness of the constellation and link assignments. A typical FOV for LEO satellites using an electro-optical communications link is usually less than 2°; however, the coarse targeting mechanism typically has an

FOV of up to 45°. In this study, the FOV was fixed while the number of effected satellites (and links) was obtained as a function of the satellite constellation. The FOV was then changed and the process repeated to determine the secondary effects of FOV on the number of effected satellites. A family of curves is obtained, with satellite position, noise position, and FOV as the input parameters.

Performance Metric

A number of metrics were considered as candidates for the performance analysis of the LEO network. These metric include, but are not limited to: effected satellites, relative edge density, per satellite link availability, k-disjoint paths, streets of coverage, and maximum revisit time. It is beyond the scope of this treatise to detail the advantages and disadvantages of these metrics. The effected satellites (or links) as a function of FOV was deemed to provide the most insight into the problem area and was chosen as the performance metric to be used in the analysis.

The effected satellites (or links) metric displays a family of curves for the percentage of satellites (or links) effected by background radiation for a particular FOV. The algorithm examines all coordinate pairs of the Sun and Moon for successive intervals through the satellite constellation's period. For each interval, the maximum, minimum and average number of satellites effected by noise is recorded. The maximum provides the worst possible impact caused by the noise sources. The minimum provides the least possible impact. The average is computed by accumulating the number of satellites (or links) effected for each Sun and Moon coordinate pair and being divided by the total number of satellites in the constellation times the number of coordinate pairs examined.

Reduction of State Space

To compute the metric it is necessary to determine which satellites are occluded by solar and lunar interference according to the formula provided in Equation 8 over the state space (every possible position of the Sun, Moon and satellite constellation). This search space may be considered a grid of Sun and Moon coordinates and a sequence of positions for the satellite constellation corresponding to their movement around a stationary Earth. The angular resolution for this analysis will be determined by the apparent

angle of the Sun and the FOV of the satellites. To identify all instances in which occlusion occurs, it is necessary to ensure that positional grid be smaller than either of these angles and that the angle between two successive line projections for any pair of satellites also be smaller than these angles. For example for a resolution (r) of 0.5 in Iridium, without state space reduction, this yields a search space[1] of

$$\left(\frac{360}{r} * \frac{180}{r}\right)^2 \left(\frac{360}{r}\right) \tag{1}$$

or 48.3×10^{12} states, clearly an unmanageable value. Fortunately there are several simplifications that may be made resulting in a reduced search grid, as illustrated in Figure 3. First the solar/lunar grid may be reduced based on astrodynamics [2]. In particular, Earth's equator is inclined to the ecliptic (the plane of the Earth's orbit) by 23°27'. Coordinates above (and below) this need not be considered. The Moon is inclined to the ecliptic between 4°59' and 5°18' thus the maximum inclination of the Moon is 28°35'. Thus we need only consider latitudes restricted to ± these values from the Equator (which we relax to 27° and 30° for the Sun and Moon respectively). For Iridium additional reductions are possible. From symmetry based on the single axis of rotation about the pole, we may ignore the search space below the equator as equivalent to the search space above. By symmetry of the number of planes, we need only consider 1/7th of half of the longitude (= 27°) to capture all the pertinent information. Finally, by symmetry on the number of satellites per plane, we need only consider 1/11th of Iridium's period (= 33°) to complete the search space. Thus the search space is reduced to

$$\left(\frac{27}{r} * \frac{27}{r}\right)\left(\frac{27}{r} * \frac{30}{r}\right)\left(\frac{33}{r}\right) \tag{2}$$

which yields 623.5×10^6 states. Notice also that the Sun and Moon are independent of one another, thus results may be obtained for the effects of only the Sun with a search grid of

---

1 360° for the longitudal position of the Sun (Moon), 180° for the latitudinal position of the Sun (Moon), and 360° for the satellites rotation.

$$\left(\frac{27}{r} * \frac{27}{r}\right)\left(\frac{33}{r}\right) \tag{3}$$

yielding only 192.5×10^3 states. For each point in the search space the number of effected satellites is computed and the cumulative results stored. This procedure is then repeated for differing values of FOV. The general form is therefore

$$\left(\frac{27}{r} * \frac{27}{r}\right)\left(\frac{D}{r}\right) \tag{4}$$

where D and r are parameters that could be changed according to need.

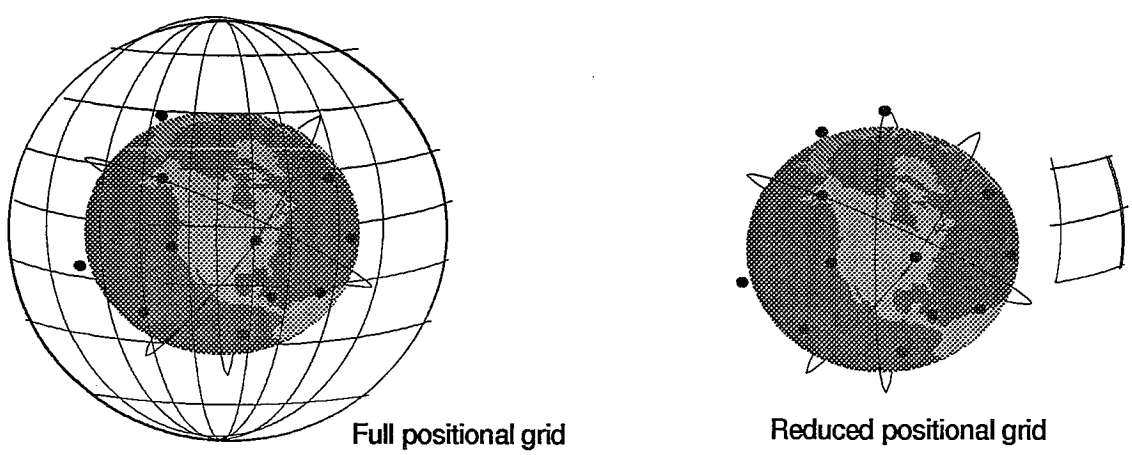

Figure 3 Solar and Lunar interference based on astrodynamic considerations.

4 SIMULATION PROCEDURE

Computation of Metric

The coordinate system used in this analysis is a geocentric equatorial coordinate system, that is the unit vectors $\left(\vec{I}, \vec{J}, \vec{K}\right)$ correspond to the direction of the Vernal Equinox, 90° eastwards, and the direction of the North Pole. The Earth rotates with respect to this coordinate system, spinning about the $\vec{K}$ axis. The

Sun and Moon are also mapped to this coordinate system located at $S = (\theta_S, \phi_S)$ and $M = (\theta_M, \phi_M)$ respectively and are assumed to subtend a solid angle when viewed from the Earth. Calculation of occlusion may be performed for both the Sun and Moon separately.

In this report only the effect of the Sun was considered. It became clear during analysis that the apparent angle of the Moon (0.5°) implies that the results for the Sun are valid for the Moon also. The combined effect of these sources is found by linear addition of the two cases. This is discussed in more detail the conclusions.

Considering only the Sun, a link between satellites $s_1 = (\theta_{s_1}, \phi_{s_1})$ and $s_2 = (\theta_{s_2}, \phi_{s_2})$ will be disrupted if any portion of the Sun is within the FOV of the line projection between these two points.

The angle subtended by the Sun from the Earth (satellite altitudes are considered negligeable) is approximately 0.5347°. If the FOV of the receiver is f then we say the link is disrupted if the angle between the furthest satellite to the Sun and the line projection from the two satellites θ (see figure 4) is less than

$$\frac{(f + 0.5347)}{2} \tag{5}$$

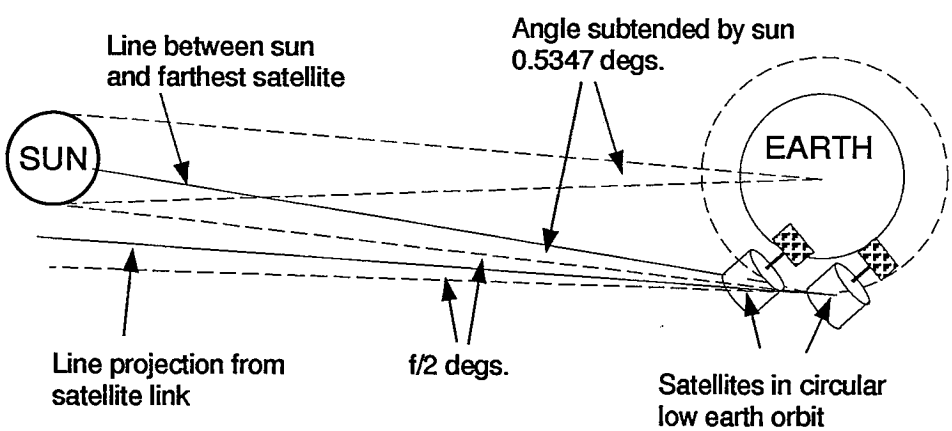

Figure 4 Occlusion between satellites due to the Sun.

The angle between its communicating satellite and Sun can be obtained using the *scalar product* which is

$$\mathbf{ab} = |\mathbf{a}||\mathbf{b}| \cos\theta \tag{6}$$

as shown in figure 5.

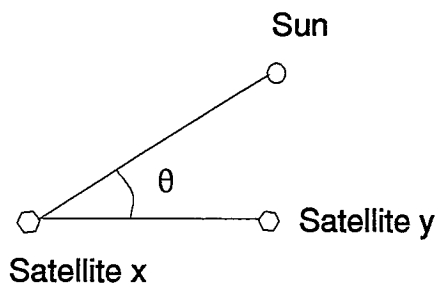

Figure 5 The angle between a pair of satellites and Sun .

Specifically, for each satellite pair (s_1, s_2):

1. Select satellite farthest from Sun

$$\begin{aligned} &\text{If } (|S - s_1| > |S - s_2|) \text{ then} \\ &\vec{A} = S - s_1,\ \vec{B} = s_2 - s_1 \\ &\text{else} \\ &\vec{A} = S - s_2,\ \vec{B} = s_1 - s_2 \end{aligned} \tag{7}$$

   Compute angle of interference

$$\theta = \arccos\left(\frac{\vec{A} \cdot \vec{B}}{\left|\vec{A}\right|\left|\vec{B}\right|}\right) \tag{8}$$

2. If $\theta < \left(\frac{f+0.5347}{2}\right)$ then link is occluded.

Simulation Procedure

Computation of the metric was accomplished using the OPNET[2] network simulation package. With this tool a simulation model may be developed as a network of connected satellites. The satellites are modelled as a network of connected process models and the process models are coded in 'C'. In addition, OPNET permits one to access models external to the program. This accomplished via the External Model Access (EMA) package, which can be viewed as an Application Program Interface (API) for creating or

[2] OPNET–B Version 2.3, Mil-3 Inc., Washington, D.C. 20008

extracting data from OPNET model files. This technique was adopted in order to improve the simulation runtime. The complete simulation sequence may therefore be viewed as process of refinement as shown in Figure 6. Once the results are obtained they may be displayed and manipulated via analysis tools provided with OPNET.

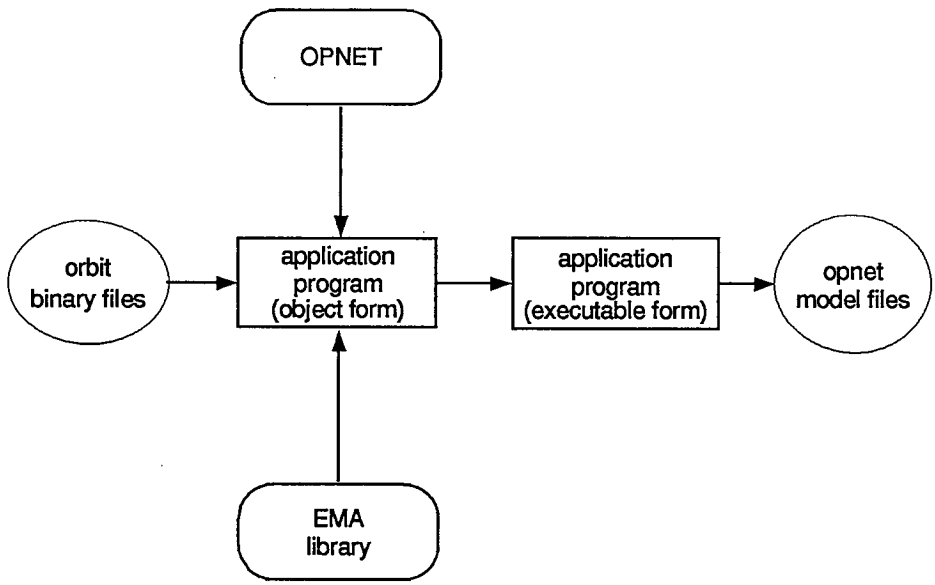

Figure 6 The application sequence

The genorb utility of OPNET was used to generate an orbit coordinate dataset for each satellite based on a specified set of orbital elements. For example, in the case of MSS, genorb is used to generate 240 binary orbit files, one for each satellite. The binary files contain global coordinates for each time step (i.e. 10 seconds) over the simulation period.

The simulation proceeds by reading in the binary orbit files to obtain the coordinates of each satellites for each simulation time; computing the angle between communicating satellites and the Sun for the current time and Sun position; accumulating the results and then updating the Sun position. This process is repeated for every time interval over the simulation period. Pseudocode for the algorithm is provided below:

```
Read_Orbit_Files();
Initialize_Statistics();
FOR (time=start;time<end;time+=time_step) {
  IF (Update_Links(time))
      Update_Satellite_Position (time);
  FOR (sun_theta=theta_min;sun_theta<theta_max; sun_theta+=theta_step)
    FOR (sun_phi=phi_min; sun_phi<phi_max; sun_phi+=phi_step) {
      FOR (sat_x=1;sat_x< TOTAL SATS;sat_x++)
        FOR (sat_y=sat_x+1;sat_y < TOTAL SATS;sat_y++)
           IF (Line_of_Sight(sat_x,sat_y)< ((f+0.5347)/2))
               Update_Statistics(sat_x,sat_y);
      Report_Statistics();
    }
}
Close_Statistics();
Close_Orbit_Files();
```

Iridium considerations The simulation time used was 24000 seconds even though the period of the system is only 6000 seconds. This was done to ascertain whether constellation periodicity is discernible in the results. The link assignments are fixed over time so they only need to be assigned once, i.e., when program begins to execute. The update of statistics is 100 seconds instead of the time step (10 seconds) to reduce computing space and is sufficient to obtain accurate results.

MSS considerations The period of MSS is close to that of Iridium, 6000 seconds. The simulation time used was 14000 seconds (approximately two times the period). Because maximum cross-links distance is limited to 1200 n. mi. and each satellite may be connected to at most six others. Link assignments are updated every 100 seconds. The algorithm proceeds as follows. The distances between all satellite pairs are computed and stored in a vector. This is sorted into increasing distance. The links are then assigned to the satellites until each satellite has as most six links under 1200 n. mi. Evaluation of this algorithm indicated that almost all satellites were able to connect to six neighbors and none were connected to less than five.

5 RESULTS

To reiterate, the results provided below are for the Sun only but are considered valid for the Moon. All graphs are plots obtained from running the simulation program discussed in Section 4.

The Iridium Constellation

Figure 7 shows the maximum, minimum and average number of satellites effected due to the solar radiation over the simulation time of 6000 seconds (corresponding to one period for the constellation) using a FOV of 20°. The value of 20° was selected for illustration purposes only. In fact, data were accumulated for each FOV from 1° to 45°. Satellite motion resolution was set at 10 seconds, corresponding to satellite angular motion of approximately 0.6°. This is well within the angular resolution of the noise source grid of 1°.

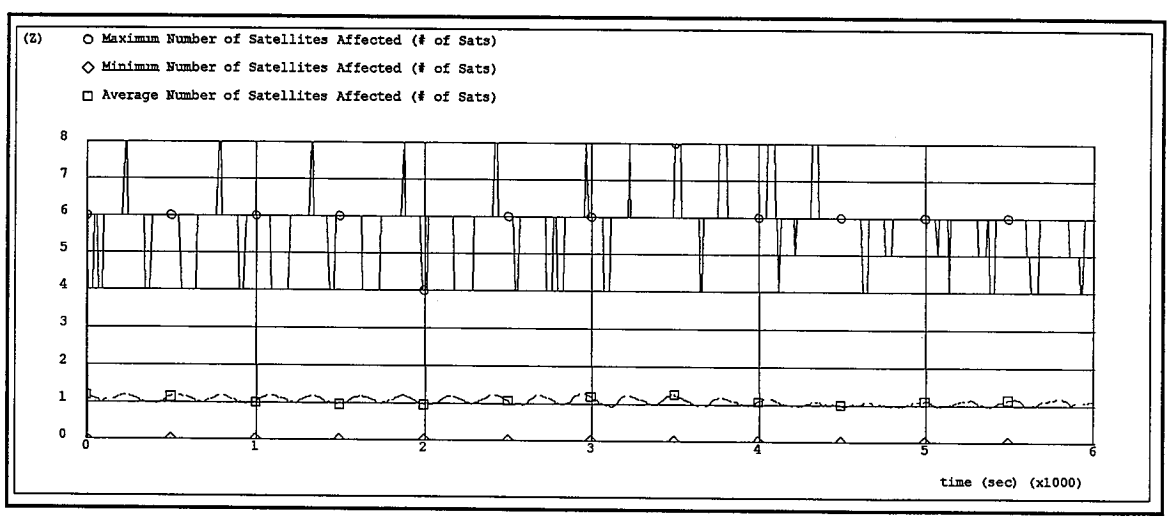

Figure 7 Maximum, average, and minium number of satellites effected vs. time, FOV = 20°

Figure 8 shows equivalent results plotted for links. Notice that the number of links effected is always equal to half the number of satellites effected. This implies that the number of satellites with two or more links broken is negligible.

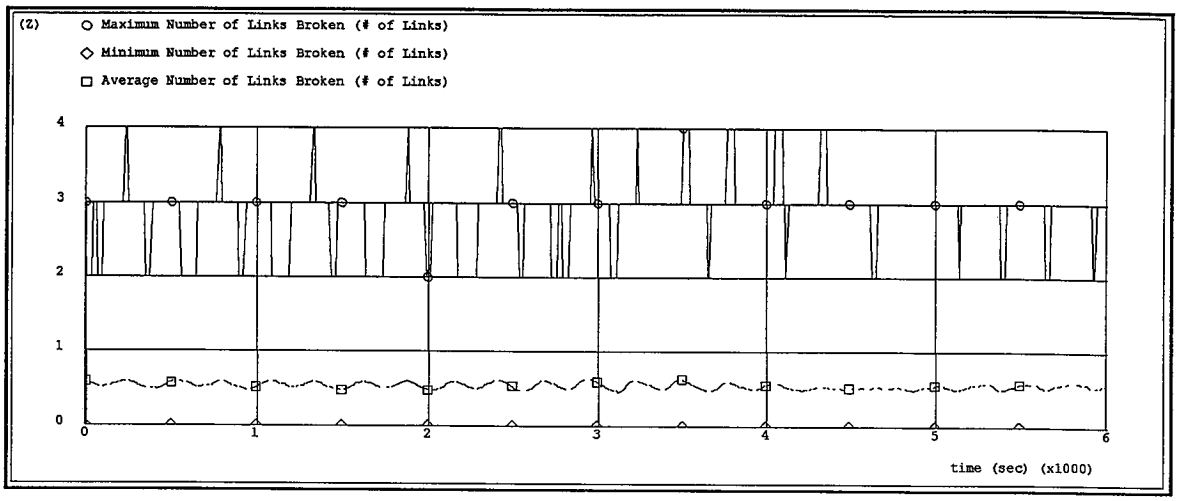

Figure 8 Maximum, average, and minium number of links effected vs. time, FOV = 20°

Figures 9 and 10 are plots reflecting the number of satellites and links effected when the FOV is varied from 1° to 45° in increments of 1°. The upper and lower bounds indicate the maximum and minimum possible effected satellites at any point in the simulation over the sample period. The graphs were obtained by post simulations analysis of the original datasets. Figure 9 is derived from 45 individual plots of number of satellites effected. In each case the sample mean of the maximum number of satellites effected was calculated by simulation over 24000 seconds using a resolution of 10 seconds. This corresponds to four complete revolutions of the Iridium system.

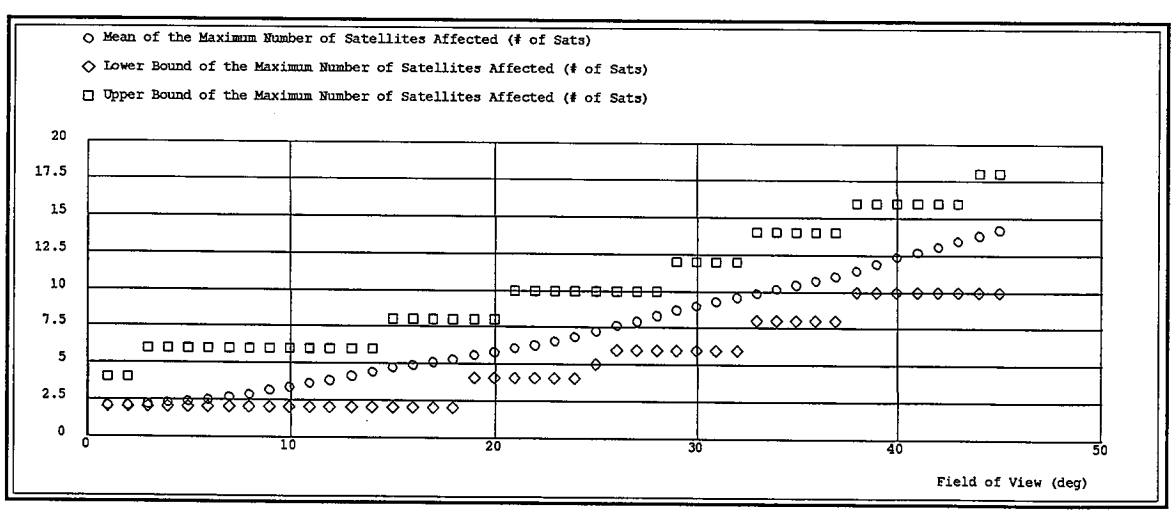

Figure 9 Maximum and minium bounds and average for the maximum number of satellites effected vs. FOV.

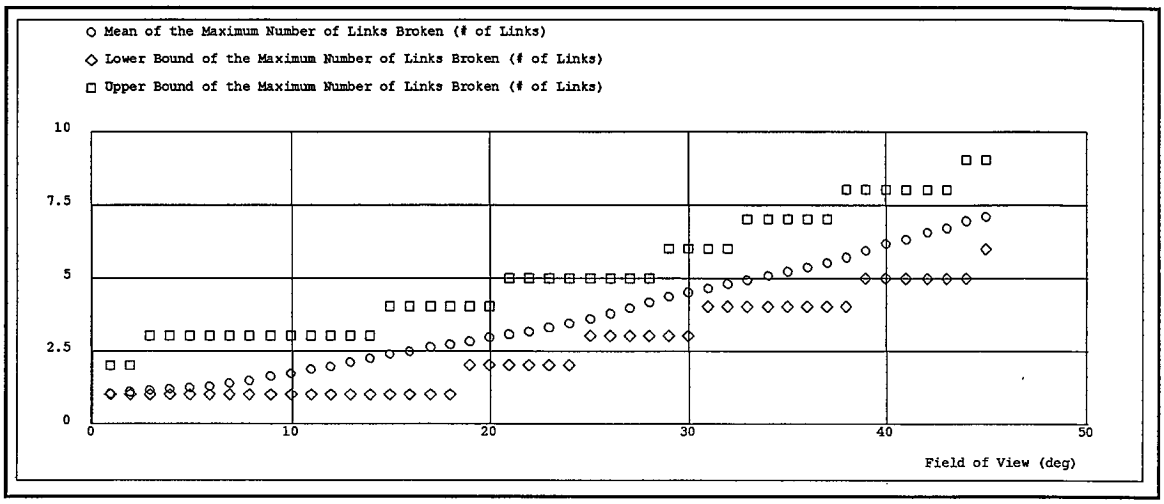

Figure 10 Maximum and minium bounds and average for the maximum number of links effected vs. FOV.

Figure 10 is a similar statistic for links between satellites. The importance of these statistics is that they provide an insight into the upper bound and averages of the effect of solar and lunar radiation on satellite connectivity as the FOV changes. This is a worst case statistic and assumes that at every point in time, the radiation effects the maximum number of satellites.

The MSS Constellation

The background for MSS is similar to that of Iridium except that constellation and the simulation time are changed. In the case of MSS there are 240 orbital files to be generated. A simulation time of 14000 second was selected with an update interval of 10 seconds. This corresponds to slightly over two complete revolutions of the MSS system.

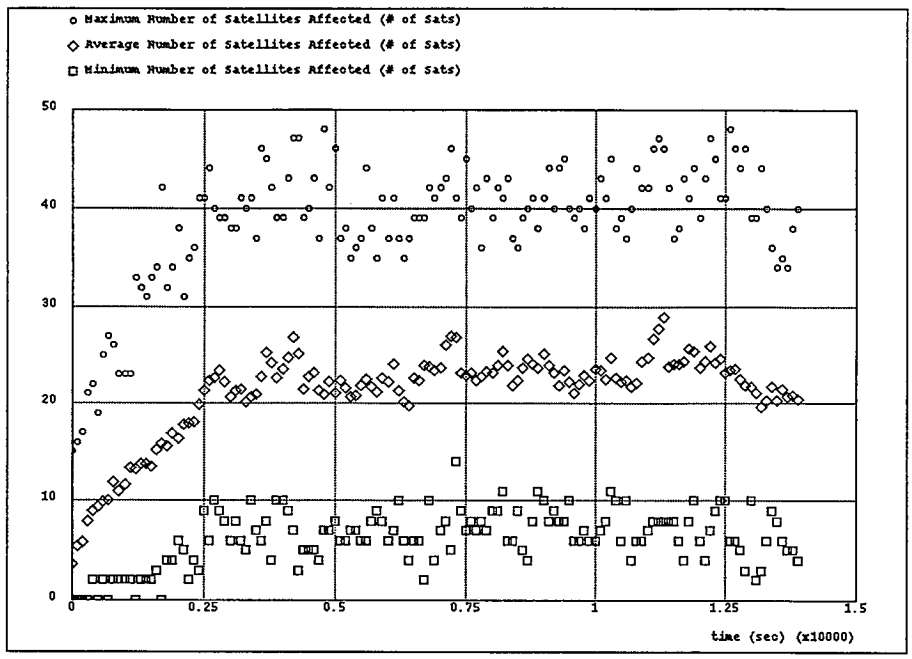

Figure 11 Maximum, average, and minium number of satellites effected vs. time, FOV = 20°

Figure 11 shows the results plotted for the number of satellites effected as a function of simulation time using a FOV of 20°. The output statistics for the graph were generated every 100 seconds.

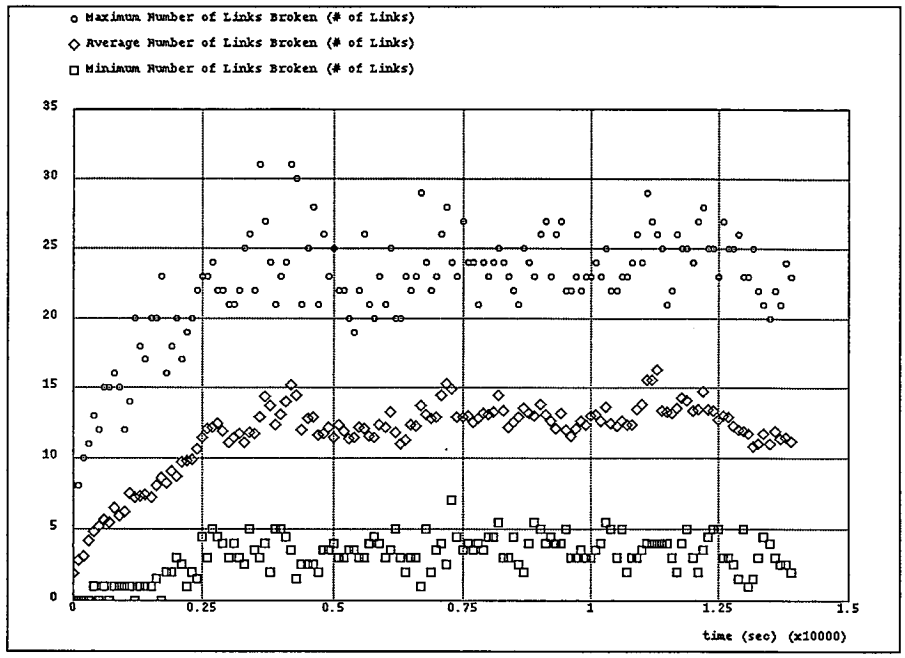

Figure 12 Maximum, average, and minium number of links broken vs. time, FOV = 20°

Compared with figure 11, figure 12 show the maximum, minimum and average number of links broken.

Figure 13 and figure 14 are post-simulation analysis from 45 original simulation results and illustrate the upper, lower, and mean bounds at different fields of view varied from 1° to 45°.

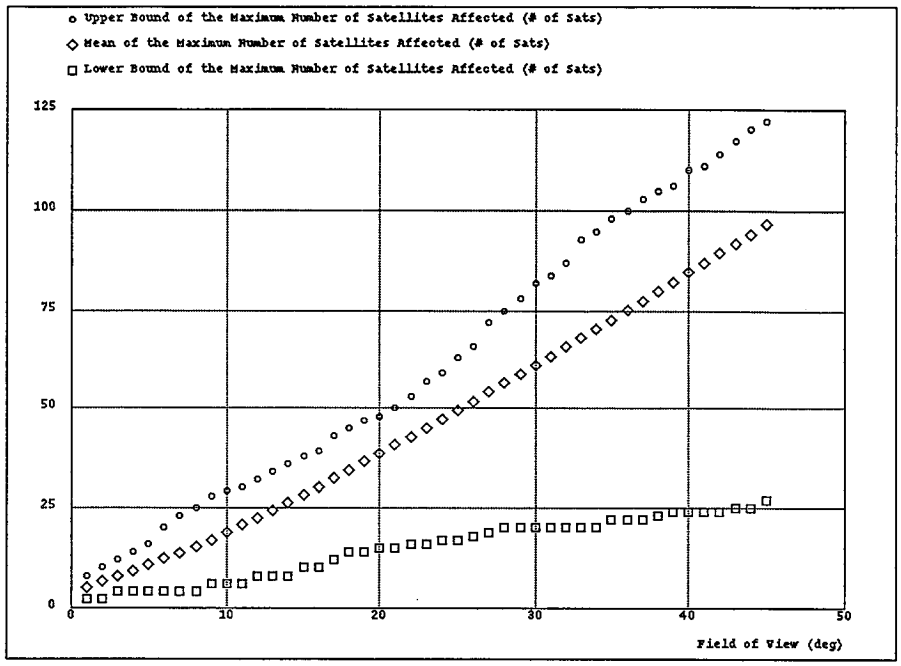

Figure 13 Maximum and minium bounds and average for the maximum number of satellites affected vs. FOV.

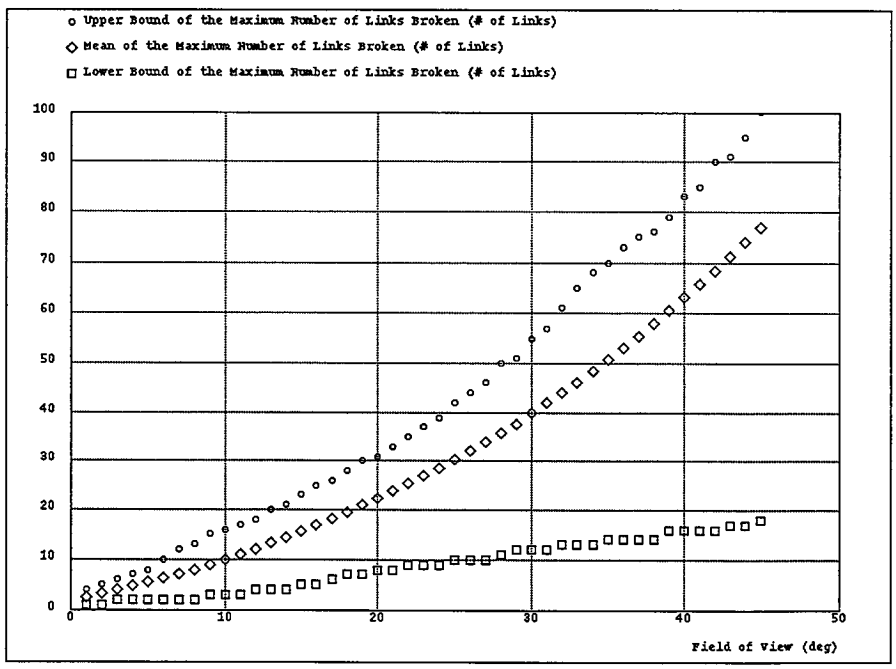

Figure 14 Maximum and minium bounds and average for the maximum number of links effected vs. FOV.

There is an interesting puzzle that needs addressing here. From figure 11 and figure 12 we see a steep gradient associated with the curves during the first 2,000 seconds followed by a leveling off for the remainder of the simulation. The reason behind this initial incline is still unclear. We would expect that the number of links effected for a system with a "random" constellation would be largely independent of time (as is the case greater than 2,000 seconds). One possible explanation for this is that the distribution of satellites deployed by the random algorithm is not initially uniform. One would expect that if we ignore accumulating the first portion of simulation time we should get more accurate results. The next two graphs (figure 15 and figure 16) showing the maximum, minimum and mean bounds on satellites (links) effected as a function of FOV are derived in this way. The chosen time range is from 2,000 seconds to 14,000 seconds.

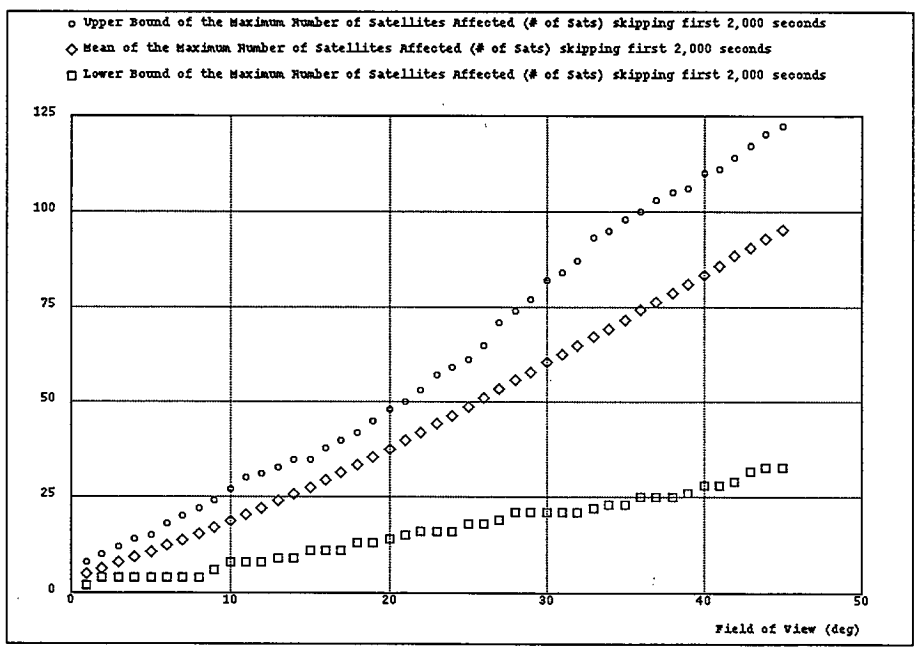

Figure 15 Maximum and minium bounds and average for the maximum number of satellites effected vs. FOV using data accumulated from 2000 seconds to 14,000 seconds.

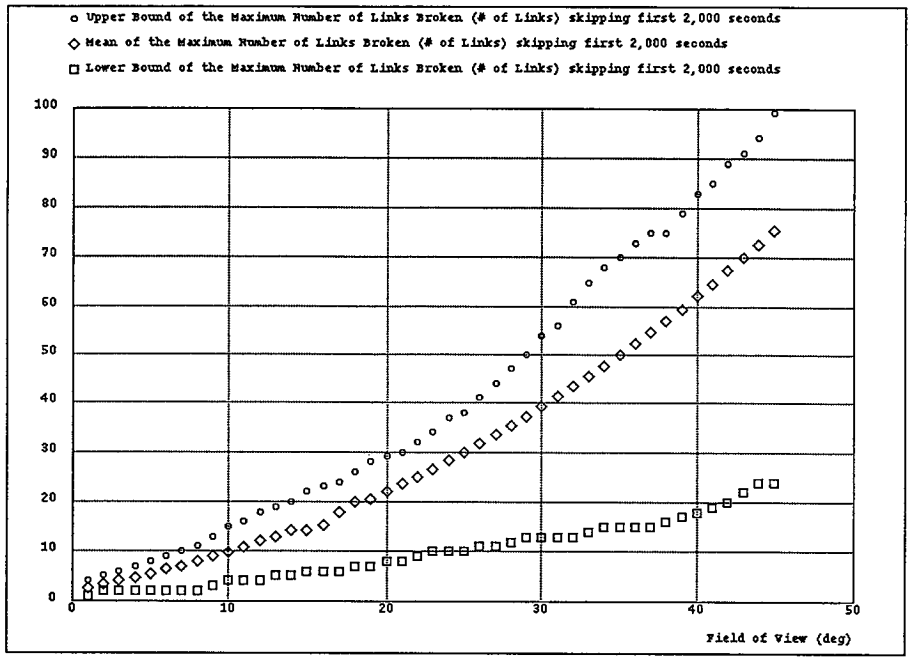

Figure 16 Maximum and minium bounds and average for the maximum number of links effected vs. FOV using data accumulated from 2000 seconds to 14,000 seconds.

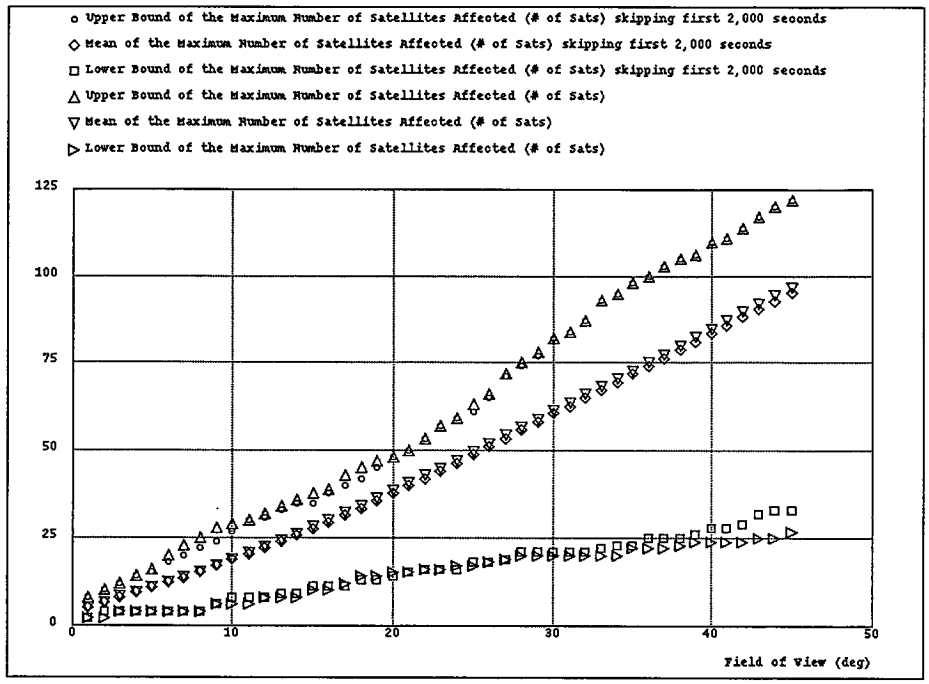

Figure 17 Superposition of figures 13 and 15. This graph shows the difference between data accumulated from 0 to 14000 seconds and 2000 to 14000 seconds.

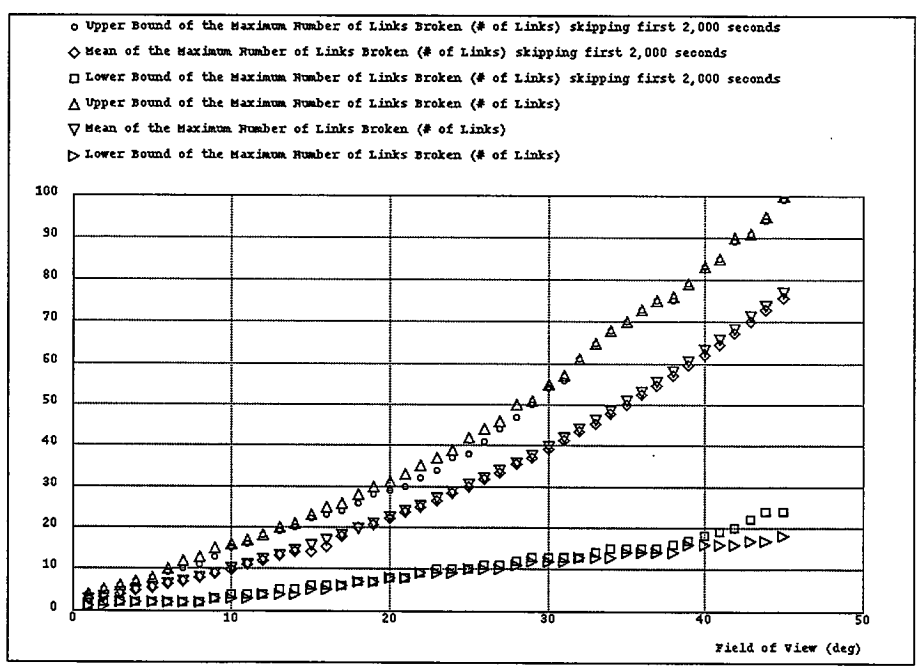

Figure 18 Superposition of figures 14 and 16. This graph shows the difference between data accumulated from 0 to 14000 seconds and 2000 to 14000 seconds.

Figure 17 compares 13 and 15 to see the impact of first 2000 seconds. Figure 18 shows equivalent results plotted for links. Figures 17 and 18 indicate no significant difference.

Comparing Iridium and MSS

It is instructive to compare the results of Iridium with those of MSS. One might expect to see significant differences between the two since once is a "regular" constellation (Iridium) and the other is a "random" constellation (MSS). Clearly with one constellation possessing 77 satellite and the other possessing 240 some normalization process is necessary before a comparison can be made.

In comparing the total numbers links assigned we note that the regular mesh of Iridium demands 143 links[3]. The average number of links per satellite is therefore 2*143/77 = 3.71. For MSS, the algorithm requires that each satellite be connected to at most 6 adjacent satellites subject to maximum length limitations. The impact of these limitations may be seen by considering the actual number of links assigned at each simulation time interval. These are illustrated in Figure 19. The majority of values are in the range 660 to 680. Using a value of 670, the average number of links per satellite is therefore 5.58 (2*670/240). The random distribution of satellites leads us to believe this value would be similar for other constellations with the same general characteristics.

---

[3] For every plane there are 11 links, thus for 7 planes the number will be 77. Also, each satellite has links with its two neighboring planes except between the first and last planes. This results in an additional 66 links. Thus, the total is 143 (i.e. 77+66= 143).

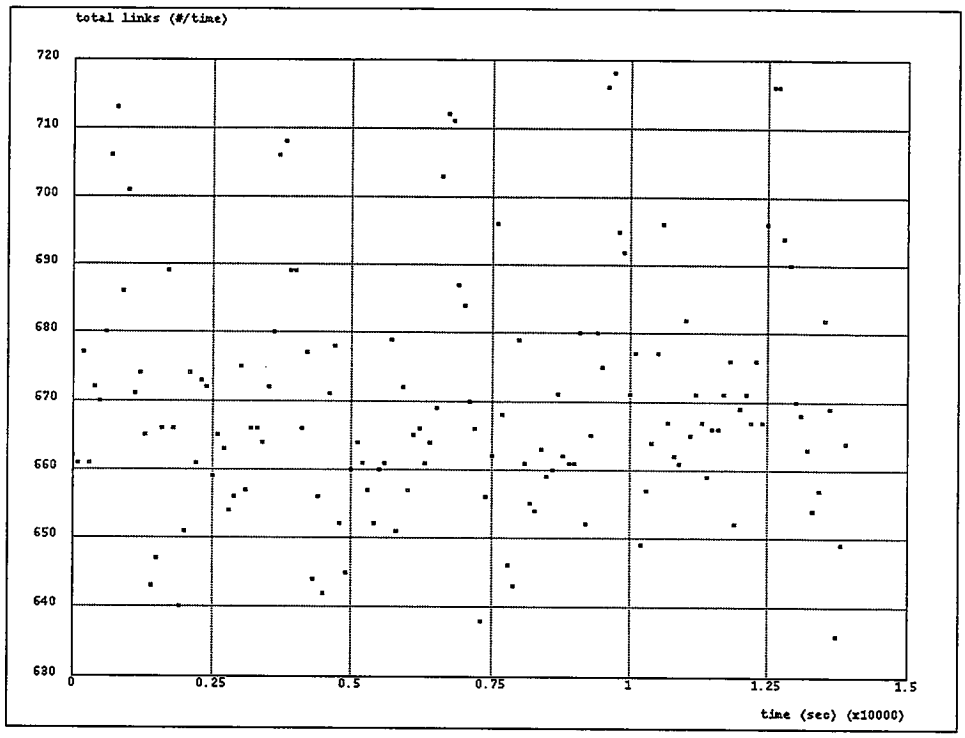

Figure 19 The total number of links assigned vs simulation time for MSS.

Figure 20 (21) shows the result of dividing the maximum number of satellites (links) effected by the total number of satellites (links) as a function of FOV for both Iridium and MSS, i.e. the number of satellites (links) effected per satellite (link). The value of 670 as an average for total links was used for MSS.

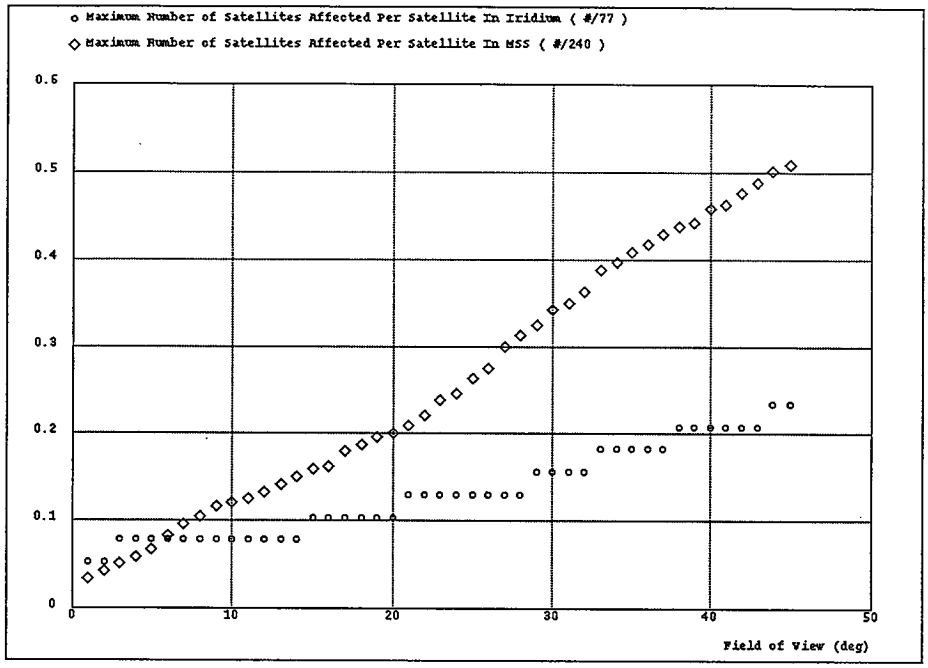

Figure 20 After normalizing the maximum satellites effected this shows the mean number of satellite effected per satellite assuming the worse case.

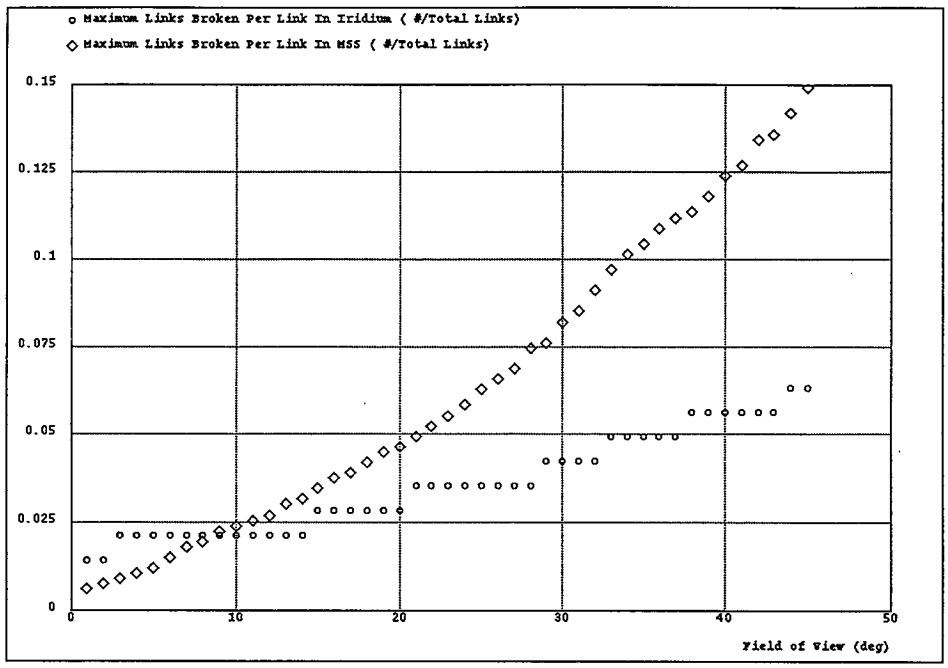

Figure 21 After normalizing the maximum links broken this shows the mean number of a links effected per link assuming the worse case.

Notice that the gradient between the two plots in figure 21 is approximately 1.6. This is completely accommodated by the ratio of the number of links per satellite in MSS over that of Iridium, i.e. 5.58/3.71 or 1.5.

Figure 22 (23) shows the ratio of the maximum number of satellites (links) effected per satellite (link) for MSS over that of Iridium.

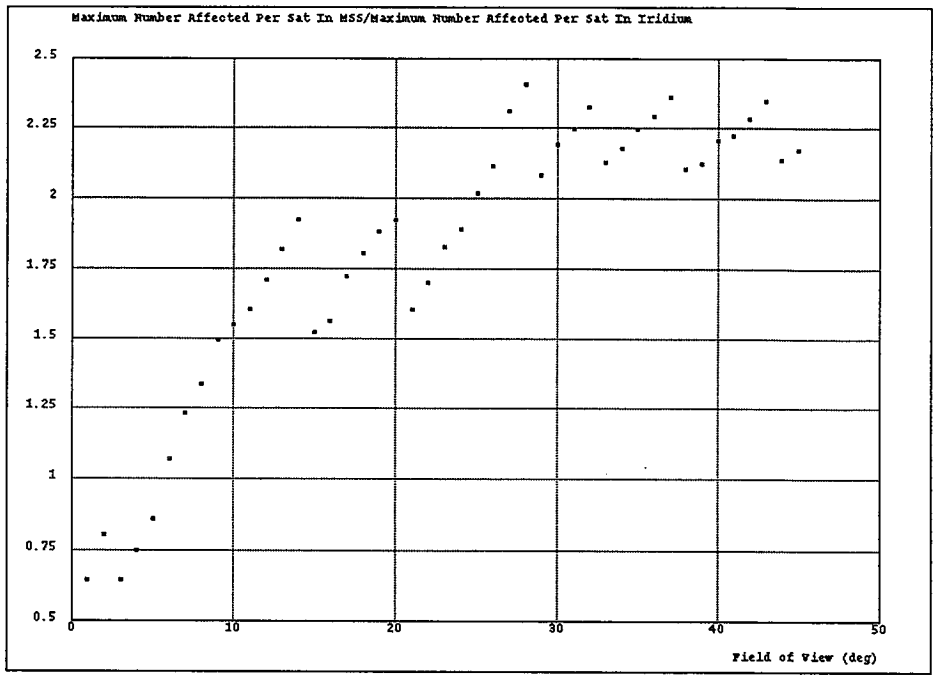

Figure 22 Ratio of satellites effected for MSS and Iridium for different FOV.

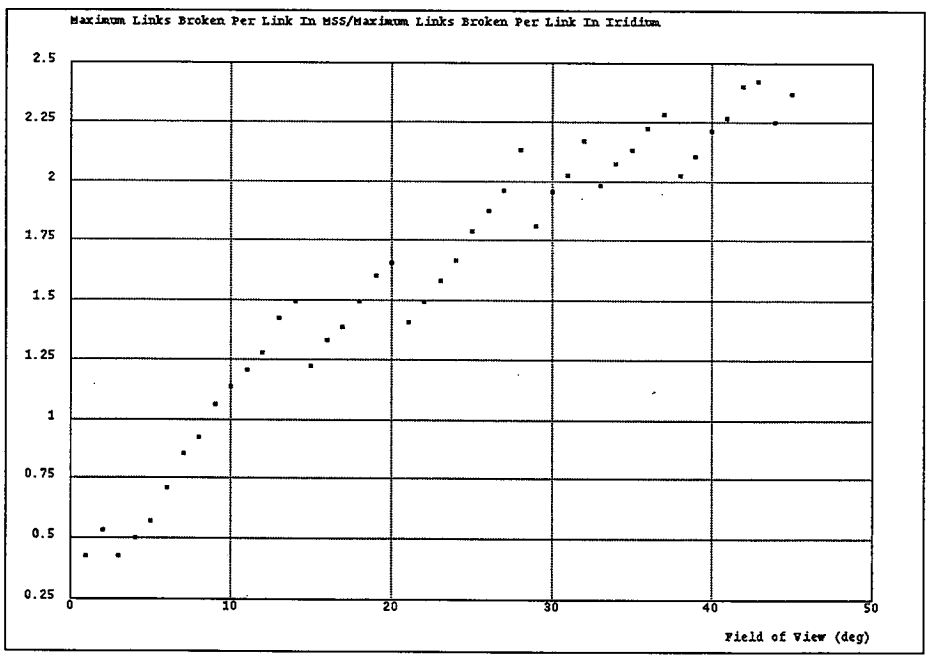

Figure 23 Ratio of links broken for MSS and Iridium for different FOV.

6 CONCLUSIONS

This report has introduced the concepts involved with analyzing the effects of solar (and lunar) radiation on laser cross-links for two low earth orbit (LEO) satellite constellations, Iridium and MSS. An number of observations may be made based on the material presented. From figures 9, 10, 15, and 16 it is clear that for small FOV values the impact of solar radiation will be negligible. E.g. the maximum number of links effected for Iridium using an FOV of 1° is only two. Even for large FOV values the impact is modest. Figures 22 and 23 suggest that with smaller FOV, Iridium is rather more susceptible to satellite (link) impact than is MSS. This is to be expected due to the regularity of Iridium. However, once the FOV reaches a "large" value (i.e. greater than ten degrees) the number of links possessed by MSS satellites as opposed to Iridium (approximately 5.58 to 3.71) increases the likelihood of some impact.

Some questions still remain as a result of this research. First, it is unclear as to why there is an initial gradient associated with measurements of satellites (and links) effected as a function of time for MSS (see figures 11, 12). It is conjectured that this is a result of a non random initial deployment of the satellites, however, this needs further study. Second, a moving average of 1000 seconds applied to the data for the average number of satellites effected in the Iridium system for FOVs of 1° and 10° shows clearly a strong correlation between them which is not clear from the original data. This is to be expected since the system is deterministic. A mathematical expression yielding satellites (links) effected as a function of FOV given a basis dataset should be attainable.

Finally, the combined impact of the Sun and Moon have not been examined. With the approach taken in this report (tuple space vs orbital elements) analysis of the two are independent. In fact, from initial results we concluded that the lunar impact is identical to the of the Sun. The combined effect on the maximum and minimum bounds may therefore be determined by the doubling the values reported here.

BIBLIOGRAPHY

[1] W. Adams and L. Rider. Circular polar constellations providing continious single or multiple coverage above a specified latitude. *The Journal of the Astronomical Sciences*, XXXV(2):155–92, 1987.

[2] R. R. Bate, D. D. Mueller, and J. E. White. *Fundamentals of Astrodynamics*. Dover Publications, New York, NY, 1971.

[3] B. R. Bertiger, P. A. Swan, and R. J. Leopold. Iridium is in the works. *Aerospace America*, 1991.

[4] R. Binder, S. D. Huffman, I. Gurantz, and P. A. Vena. Crosslink architecture for a multiple satellite system. *Proceedings of the IEEE*, 75(1):74–82, 1987.

[5] P. S. Jorgensen. Navstar/global positioning system 18-satellite constellations. *Journal of the Institute of Navigation*, 27(2):89–96, Summer 1980.

[6] J. Lehman and et al. MSS requirements and performance analysis. Technical Report ESL-TM-1632, ESL Inc., Sunnyvale, CA, 1983.

[7] B. W. Parkinson and S. W. Gilbert. Navstar: Global positioning system-ten years later. *IEEE Transactions on Communications*, COM-31(10):1177–1186, 1983.

[8] C. Pentlicki. Multiple satellite system program/cost projection and evaluation. F30602-96-C-0063 E598, Comsat Laboratories, Clarksburg, MD, 1987.

[9] M. Richharia, P. H. Hansel, P. W. Bousquet, and M. O'Donnell. A feasibility study of a mobile communication network using a constellation of low earth orbit satellites. In *Globcom'89*, pages 773–7, Dallas, TX, 1989.

Appendix A Source Code to Generate Orbit Files

```
/*****************************************************************************/
/* The program generate orbit binary file. The lon_ascen and mean_anomaly
parameters will be generate by random(). The inclin angles will be set according
documents to  27.5, 57.5 , and 90 degree three  groups. Every group have 80
member orbit.
        The lon_ascen will be unique in every group, becase it will be
insert to binary tree to make sure there is no duplicate. The mean_anomalys
do't have such limitation, becasue after making sure lon_ascen is different
there will be unique orbit in every 80 members group .
        The whole orbit parameters can be showed by using "vieworb" utility.
The format is something like "vieworb mod_dirs PATH -m FILE". The FILE do't
need the "orb" extention.
        Need to compile with bin_main.c, becasue we use it's function. */
/*****************************************************************************/

#include <stdio.h>
#include "tree.h"
#include <math.h>

extern   struct node  *search(struct node *root, double value);
extern   void insert( struct node *root, double number);
extern   void print(struct node *node_pt);

FILE     *f,*g ;

extern   struct  node
       {
         double       key;
         struct node  *left,
                      *right;
       } ;


extern  struct node  *head, *result ;


 main()
    {
      char     buffer[500];
      int      i,n, j ;
      char     start_date[20],start_time[20], stop_date[20], stop_time[20],
               time_step[20],
               waste[20];    /* read in the unusefull data */
      char     check_point[3];
      char     elem_type[20],eccen[20],
               tolerance[20],drag_area[20],mass[20],drag_coeff[20];
      double   mean_anomaly;
      double   lon_ascen,max_alt,sm_axis,range_sm_axis;
      double   min_sm_axis;
      int      count=1;
      double   incline;



      if ( (f = fopen("m4_orbit_data.x", "r")) == NULL )
           {
              printf("m4_orbit_data.x file can't be opened\n");
              exit(-1);
            }


     if ( (g=fopen("m4_orb_data_output.x", "w")) == NULL)
          {
              printf("Have problem to open m4_orb_data.x \n");
              exit(-1);
            }

      fscanf(f,"%s %s", waste, start_date);
      fscanf(f,"%s %s", waste, start_time);
      fscanf(f,"%s %s", waste, stop_date);
      fscanf(f,"%s %s", waste, stop_time);
      fscanf(f,"%s %s", waste, time_step); /*unit is seconds */
      fscanf(f,"%s %s", check_point,waste);
      if ( strcmp(check_point,"*") != 0)
            {
              printf("Error at check point '*'\n");
              return;
             }

          fscanf(f,"%s %s", waste, elem_type);
          fscanf(f,"%s %s", waste, eccen);
  /*        fscanf(f,"%s %s", waste, incline);  */
          fscanf(f,"%s %s", check_point,waste);


      if ( strcmp(check_point,"&") != 0)
            {
              printf("Error at check point '&'\n");
              return;
             }

          fscanf(f,"%s %s", waste, tolerance);
          /* fscanf(f,"%s %s", waste, max_alt);*/
          fscanf(f,"%s %s", waste, drag_area);
          fscanf(f,"%s %s", waste, drag_coeff);
          fscanf(f,"%s %s", waste,  mass);
          fscanf(f,"%s %s", check_point,waste);
      if ( strcmp(check_point,"%") != 0)
            {
              printf("Error at check point '%'\n");
              return;
             }

  max_alt = 740.8;
/* Maximum altitude. 400 n.mi +Earth radium = 400 * 1852 + 6.378e6 = 7118.8
                     or 400*1852= 740800  */

  min_sm_axis = 350.0*1852+ 6.378e6; /* altitude is from 350 n.mi to 400 n.mi */
  range_sm_axis = 50.0 *1852;      /* range is 50 n. mi */


for(i=1;i<4 ;i++) {
  count = 0;
  head= NULL;

  switch (i) {
  case  1 :  incline = 27.5 ;
             break;
  case  2:   incline = 57.5 ;
             break;
  case  3:   incline = 90.0 ;
             break;
  default:   printf("\nwrong  incline angle assigned");
             exit(-1);
    }

  fprintf(g,"Generate %i group orbit \n",i);
```

```
do {
     mean_anomaly = random()/UNIT*DEGREE;
     sm_axis = (min_sm_axis + range_sm_axis *random()/UNIT)/1000.;
     fprintf(g,"count=%i\n",count);
     fprintf(g,"nmean_anomaly=%f\nsm_axis      =%f\n",\
                                  mean_anomaly,sm_axis);

   do {
        lon_ascen = (int) (random()/UNIT*DEGREE);
        fprintf(g,"lon_ascen    =%f\n",lon_ascen);
        result= search(head,lon_ascen);
        if( result== NULL)
           if (head==NULL){
                 head = (struct node *) malloc(sizeof(struct node));
                 head->key = lon_ascen;
                 head->left= NULL;
                 head->right = NULL;
                 count++;
           }
           else
           {
            count++;
            insert(head,lon_ascen);
           }
        else
        fprintf(g,"------ lon_ascen=%f already in tree -------\n",\
                                lon_ascen);
        fprintf(g,"\n");
    } while (result != NULL);

        sprintf(buffer, "genorb  \
                -mod_dirs /home/cse_h1/sdiproj/op_models/Test/Orbit \
                -start_date %s \
                -start_time %s -stop_date %s -stop_time %s -time_step %s \
                -elem_type %s -sm_axis %f -eccen %s -inclin %f -lon_ascen %f \
                -mean_anomaly %f -tolerence  %s -max_alt %f -drag_area %s \
                -drag_coeff %s -mass %s -orb_name  m4_orb%d_%d   ",
                 start_date,
                 start_time, stop_date, stop_time, time_step, elem_type,
                 sm_axis, eccen, incline, lon_ascen,
                 mean_anomaly, tolerance, max_alt, drag_area,
                 drag_coeff, mass, i, count);
    system(buffer);

   } while (count < SIZE ); /* end of while loop */
  fprintf(g,"\nThe incline = %f", incline );
  fprintf(g,"\nThe sorted lon_ascen are ");
  print(head);
  fprintf(g,"\n----------------------------------------\n");
 } /*end of for loop */


    printf("\n------- end of generating orbit ---------------\n");
    fprintf(g,"\n*********** End of generating orbit ************\n");
 }
```

```
 Orbit file m4_orb2_5
------------------------- Orbit Attributes --------------------------
sun_effects:                off
moon_effects:               off
drag_effects:               off
srp_effects:                off
elem_type:                  "mean"
sm_axis:                    7,052.12651
eccen:                      0.0
inclin:                     57.5
lon_ascen:                  198
arg_perigee:                0.0
mean_anomaly:               24.435291
tolerance:                  1E-09
time_step:                  10
start_date:                 "1993.6.20"
start_time:                 "12.00.00.00"
stop_date:                  "1993.6.20"
stop_time:                  "22.00.00.00"
max_alt:                    740.8
drag_area:                  0.0083
drag_coeff:                 2.1
srp_area:                   1E-06
srp_coeff:                  0.0044
mass:                       100
----------------------------------------------------------------------

 Orbit file m4_orb2_10
------------------------- Orbit Attributes --------------------------
sun_effects:                off
moon_effects:               off
drag_effects:               off
srp_effects:                off
elem_type:                  "mean"
sm_axis:                    7,111.517573
eccen:                      0.0
inclin:                     57.5
lon_ascen:                  297
arg_perigee:                0.0
mean_anomaly:               187.203151
tolerance:                  1E-09
time_step:                  10
start_date:                 "1993.6.20"
start_time:                 "12.00.00.00"
stop_date:                  "1993.6.20"
stop_time:                  "22.00.00.00"
max_alt:                    740.8
drag_area:                  0.0083
drag_coeff:                 2.1
srp_area:                   1E-06
srp_coeff:                  0.0044
mass:                       100
----------------------------------------------------------------------

 Orbit file m4_orb2_44
------------------------- Orbit Attributes --------------------------
sun_effects:                off
moon_effects:               off
drag_effects:               off
srp_effects:                off
elem_type:                  "mean"
sm_axis:                    7,042.107897
eccen:                      0.0
inclin:                     57.5
lon_ascen:                  68
arg_perigee:                0.0
mean_anomaly:               244.001871
tolerance:                  1E-09
time_step:                  10
start_date:                 "1993.6.20"
start_time:                 "12.00.00.00"
stop_date:                  "1993.6.20"
stop_time:                  "22.00.00.00"
max_alt:                    740.8
drag_area:                  0.0083
drag_coeff:                 2.1
srp_area:                   1E-06
srp_coeff:                  0.0044
mass:                       100
----------------------------------------------------------------------

 Orbit file m4_orb2_79
------------------------- Orbit Attributes --------------------------
sun_effects:                off
moon_effects:               off
drag_effects:               off
srp_effects:                off
elem_type:                  "mean"
sm_axis:                    7,032.599719
eccen:                      0.0
inclin:                     57.5
lon_ascen:                  315
arg_perigee:                0.0
mean_anomaly:               173.200567
tolerance:                  1E-09
time_step:                  10
start_date:                 "1993.6.20"
start_time:                 "12.00.00.00"
stop_date:                  "1993.6.20"
stop_time:                  "22.00.00.00"
max_alt:                    740.8
drag_area:                  0.0083
drag_coeff:                 2.1
srp_area:                   1E-06
srp_coeff:                  0.0044
mass:                       100
----------------------------------------------------------------------
```

Appendix B Source Code for Simulation

```
#include <stdio.h>
#include <math.h>
/*
#include <stdlib.h>
#include "output_model.h"  */

/*\
|*| orb.h: Header file for Opnet satellite orbit position files.
|*| See vol. 6 for details of GenOrb utility.
\*/

#define FIELD_OF_VIEW     1
#define SUN_VIEW_ANGLE    0.535
#define MOON_VIEW_ANGLE   0.516
#define SIM_TIME          70
#define INC_TIME          10
#define INC_INDEX          1


#define SUN_R             6.378E+08  /* One hundred tiems of earth radius */
#define MOON_R            4.464E+07  /* Seven times of earth radius */

#define EARTH_RADIUS      6.378E+06
#define RADIUS_RATIO      1.6329932

#define GRID_SUN_PHI      27   /* SUN grid from 161 to 161+27  */
#define GRID_SUN_THETA    30

#define PHI_INC           0.5
#define THETA_INC         0.5
#define SET_SIZE          80
#define SETS              3
#define TOTAL_SATS       (SETS*SET_SIZE)

#define MAX_LINKS         6    /* the max links assignment one sat. can have */
#define MAX_DIST          2222400 /* The max distance two linked satellites */

 typedef struct
    {
       double  latitude;
       double  longitude;
       double  altitude;
     } OrbT_Pos ;

  typedef struct
    {
      int         sun_effects;
      int         moon_effects;
      int         drag_effects;
      int         srp_effects;
      char        elem_type[128];
      double      sm_axis;
      double      eccen;
      double      inclin;
      double      lon_ascen;
      double      arg_perigee;
      double      mean_anomaly;
      double      tolerance;
      double      time_step;
      char        start_date[16];
      char        start_time[32];
      char        stop_date[16];
      char        stop_time[32];
      double      max_alt;
      double      drag_area;
      double      drag_coeff;
      double      srp_area;
      double      srp_coeff;
      double      mass;
    } OrbT_Desc;


typedef struct
   { double x,
            y,
            z ;
    } Sat_cord;

   typedef  struct
    {
      double      step;
      int         num_pos ;
      OrbT_Pos    *pos_array;
      Sat_cord    *pos2_array;
     } OrbT_Pset;


typedef struct
  {
    char         str[80];
    FILE         *fptr ;
    OrbT_Desc    *odptr;
    OrbT_Pset    *opsptr;
  } SatT_Info ;

SatT_Info  satinfo[TOTAL_SATS];
double field_of_view, half_fov, max_view_for_sun;

#define SatLat(sat,index)        (satinfo[(sat)].opsptr-> \
                                 pos_array[(index)].latitude)
#define SatLong(sat,index)       (satinfo[(sat)].opsptr-> \
                                 pos_array[(index)].longitude)
#define SatAlt(sat,index)        (satinfo[(sat)].opsptr-> \
                                 pos_array[(index)].altitude)


#define SatX(sat,index)         (satinfo[(sat)].opsptr-> \
                                      pos2_array[(index)].x)
#define SatY(sat,index)         (satinfo[(sat)].opsptr-> \
                                      pos2_array[(index)].y)
#define SatZ(sat,index)         (satinfo[(sat)].opsptr-> \
                                      pos2_array[(index)].z)

int       table[TOTAL_SATS][TOTAL_SATS];
int       time_step, index_step;
int       times_of_index_step;
char      *err_file;
FILE  *g;
```

```
/*                      Output model header file
 *
 *      This file needs to be included in the main program which uses the
 *      routines :
 *                op_stat_write_ema(.......)
 *                op_out_init_ema(....)
 *                op_out_finish_ema(.....)
 *
 */

/*      includes from OPNET     */
#include <opnet.h>
#include <ema.h>
#include <opnet_emadefs.h>
#include <opnet_constants.h>

/*      maximum number of statistics possible (can be set as required)  */
#define MAX_STATS  10

/*      external function definitons. */
extern void op_out_init_ema();
extern void op_stat_write_ema();
extern void op_stat_finish_ema();
extern void op_stat_init_write_ema();

/*      global variables */
static EmaT_Model_Id    model_id;               /* Model id */

static struct {
    int size;                                   /* max size of vector */
    int index;                                  /* current element in vector */
    EmaT_Object_Id stat_obj;                    /* object ids */

} statinfo[MAX_STATS];
```

```
#include "output_model.h"
#include "m4.h"

/*
#include <strings.h>
#include "m4_init_state.c"
#include "m4_read_sat.c"
#include "output_model.c"
#include "m4_proc.c"
#include "m4_begin_end.c"  */



/*\
|*| main(): reads in the orbital elements and position information
|*| of each of the orbit binary files specified. Call link1() to simulate.
|*| It will generate picture output to "m4_output".
\*/


extern  int   index_size;
extern  int   inc_time;
extern  int   time_step;
extern  char  *out_file,*fov_file;


void main()
{
 int     index=1, i, time;

  readsatellites();

  init_state();

  op_stat_init_ema(EMA_MODE_ERR_HALT);

  op_stat_init_write_ema(1,  index_size, "time", "Max Satellites");
  op_stat_init_write_ema(2,  index_size , "time", "Min Satellites");
  op_stat_init_write_ema(3,  index_size, "time", "Avg Satellites until now");

  op_stat_init_write_ema(4,  index_size, "time", "Max Links");
  op_stat_init_write_ema(5,  index_size , "time", "Min Links");
  op_stat_init_write_ema(6,  index_size, "time", "Avg Links until now ");

  op_stat_init_write_ema(7,  index_size, "time", "Avg sats of current time");
  op_stat_init_write_ema(8,  index_size, "time", "Avg Links of current time");


  time_step = satinfo[1].odptr->time_step ;
  index_step = inc_time/time_step;
  fprintf(stderr,"\nThe sats's time_step=%d seconds.\nindex_step=%d\n",
                          time_step,index_step);

  begin_time();

    if ( (g = fopen(out_file ,"w")) == NULL)
    {
      fprintf(stderr,"File %s  can't open\n", out_file);
      exit(-1);
     }

  for (index=0 ;index<(index_size *index_step) ;index += index_step)
   {
      time =time_step * index;
      if ((time%10) == 0 )
      printf("\nthe sim_time = %i\n", time);

      link1(index); /* the main compute satellite function .*/

    } /* end of for loop */

  op_stat_finish_ema(fov_file);

  printf("End of simulation .\n");

  end_time();

} /* end of main */
```

```
/*\
|*| m4_read_sat.c: Reads in the orbital information for a satellite
|*| constellation.  The information is stored as an array of file
|*| pointers and associated state variables.
\*/

/*\
|*| readsatellites() - ....
\*/


#include  "m4.h"



int readsatellites()
{
  int   i, j;
  int   set;

  for(set=0;set<SETS ;set++)
   {
    for(j=0;j<SET_SIZE;j++)
      {
        i= set*SET_SIZE+j;

 sprintf(satinfo[i].str,
    "/home/cse_h1/sdiproj/op_models/Test/Fov/m4_orb%d_%d.orb",set+1,
        j%SET_SIZE+1 );

        if ((i%10== 0) ||(i== SETS *SET_SIZE -1))
           printf("Reading %s ... ,i = %i ", satinfo[i].str, i );
        if( (satinfo[i].fptr =  fopen(satinfo[i].str,"r") ) == NULL)
           {

              printf("Can't open %s\n",satinfo[i].str);
              exit (-1);
            }
        /* allocate memory for the orbit descriptor */
        satinfo[i].odptr = (OrbT_Desc *)malloc (sizeof(OrbT_Desc));

        /* read in orbit descriptor */
        fread(satinfo[i].odptr, sizeof(OrbT_Desc), 1,
                 satinfo[i].fptr);

        /* allocate memory for orbit pointset */
        satinfo[i].opsptr =(OrbT_Pset *) malloc (sizeof (OrbT_Pset));

        /* read in the step and number-of-points values */
        fread(&(satinfo[i].opsptr->step), sizeof(double), 1,
                  satinfo[i].fptr);
        fread(&(satinfo[i].opsptr->num_pos), sizeof(int), 1,
                  satinfo[i].fptr);

        /* allocate memory for the points */
        satinfo[i].opsptr->pos_array = (OrbT_Pos *)
                malloc(sizeof(OrbT_Pos)*satinfo[i].opsptr->num_pos);

        satinfo[i].opsptr->pos2_array = (Sat_cord *)
                malloc(sizeof(Sat_cord)*satinfo[i].opsptr->num_pos);


        /* read in orbit array values */
        fread(satinfo[i].opsptr->pos_array, sizeof(OrbT_Pos),
                  satinfo[i].opsptr->num_pos, satinfo[i].fptr);
        if ((i%10== 0) ||(i== SETS *SET_SIZE -1))
           printf("done\n");
        if( fclose(satinfo[i].fptr) == EOF)
           {
              fprintf(stderr,"Can't close file %s\n",satinfo[i].str);
              exit(-1);
            }
      } /* end of for loop */
   } /* end of for loop */
     fprintf(stderr,"\nThe num_pos = %d", satinfo[i].opsptr->num_pos);
     return(satinfo[1].opsptr->num_pos);

} /* end of readsatellites */
```

```c
/*-----------------------------------------------------------------------*/
/* This program initilize all necessary value for being used in m3_proc.c  */
/*-----------------------------------------------------------------------*/
#include "m4.h"
#include "stdlib.h"


 char    *fov_file, *out_file;
 int     sim_time, index_size,
         inc_time, inc_index;
 char    *temp;

int  init_state()
{


 fov_file = getenv( "FOV_FILE");  /* Output file name from "FOV_FILE */

 if (fov_file==NULL)
  {
    fprintf(stderr, "\nEnvironment variable FOV_FILE not set.");
    exit(-1);
   }

temp = getenv( "FOV" ); /* Field of View value from "FOV" */
if (temp == NULL)
 {
   fprintf(stderr,"\nDefault FOV value = %d degree assumed.",FIELD_OF_VIEW);
   field_of_view = FIELD_OF_VIEW;
  }
 else
 {
   fprintf(stderr,"\nfield_of_view = %d degree.", atoi(temp));
   field_of_view = atof(temp);
  }
   half_fov = field_of_view/2;
                                /* Used at update_affected_sats */
   max_view_for_sun = (half_fov+SUN_VIEW_ANGLE)/2;


 temp=getenv("INC_TIME");
 if (temp== NULL)
   {
     fprintf(stderr,"\nDefault INC_TIME=%d seconds.", INC_TIME);
     inc_time = INC_TIME;
    }
  else
    {
     inc_time = atoi(temp);
     fprintf(stderr,"\ninc_time = %d seconds.", inc_time);
    }



  temp = getenv( "SIMTIME" );   /* Simulation time from "SIMTIME" */
 if (temp == NULL)
   {
     fprintf(stderr, "\nDefault SIMTIME=%d seconds.", SIM_TIME);
     sim_time = SIM_TIME;
     index_size = SIM_TIME/inc_time;
   }
 else
   {
     sim_time = atoi(temp);
     index_size = sim_time/inc_time;
     fprintf(stderr, "\nsim_time=%d seconds.", atoi(temp));
    }


  if ( (out_file =getenv("OUT_FILE")) == NULL )
      {
       fprintf(stderr,"\nDefaul file %s","m4_out_file");
       out_file = "m4_out_file";
       }
  else
       fprintf(stderr,"\nErr file %s", out_file );


  time_step = satinfo[1].odptr->time_step ;
 temp = getenv( "TIMES_OF_INDEX_STEP" );   /* Simulation time from "SIMTIME" */
 if (temp == NULL)
   {
     times_of_index_step = inc_time/time_step ;
     fprintf(stderr, "\nDefault TIMES_OF_INDEX_STEP=%d ",
                                  times_of_index_step);
   }
 else
   {
     times_of_index_step = atoi(temp) * (inc_time/time_step);
     fprintf(stderr,"\ntimes_of_index_step =%d ",times_of_index_step);
    }


};  /* end of init_state */
```

```
/*************************************************************************
output_model.c

        This file contains the four functions needed to create an output
 vector model for OPNET. These are not standalone functions they need to be
 used in a main program in the correct order.

        The output_model.h and output_model.c files need to be included in the
 main program. The main program should first call the op_stat_init_ema
 function.
        After calling op_stat_init_ema the function op_stat_init_write_ema
 must be called for each stat id, passing the names and id of the stat
 to be initialized.  Once initialized, the function op_stat_write_ema
 can be called at any time for that statistic.

        After all the data has been collected for the simulation the model
 can be stored in a file by passing the file name to op_stat_finish_ema.

        An example has been provided for refrence.

*************************************************************************/


#include "output_model.h"


/*
        Function: op_stat_init_ema(OPNET_CONSTANT)

        Parameters:
                EMA_MODE_ERR_HALT
                EMAC_MODE_ERR_PRINT
                EMAC_MODE_DIAG
                                Described on pg20-22 External Interfaces
                                Manual OPNET #6

        Use: This function sets up the basic model and debugging levels
        for the EMA functions.

        IMPORTANT: It needs to be called before any other EMA related function
        in the main program.

*/

void op_stat_init_ema(mode)

   int mode;
{
    /* initialize the EMA system */
    Ema_Init(mode);

    /* create the model for the output vectors */
    model_id=Ema_Model_Create(MOD_OUTVECTORS);



}


/*************************************************************************/
/*
        Function: op_stat_finish_ema(file_name)

        Parameters:
                string: type char* is a name of the file in which the
                        output vectors need to be stored.

        Use:  This should be called at the end of the simulation to
        store the model.

*/

void op_stat_finish_ema(file_name)

   char *file_name;

{
    int i;
    int error=0;

    /* Check that the correct number of vector elements were written */
    for (i=0;i<MAX_STATS;i++)
        if ( (statinfo[i].size !=0) &&
             (statinfo[i].size != (statinfo[i].index+1)) ) {
            fprintf(stderr,"op_stat_finish_ema: vector for stat id %i not filled\n",i);
            error = 1;
        }
    if (error) {
        fprintf(stderr,"op_stat_finish_ema: output not written\n");
        exit (-1);
    }

    /* write the model into a file_name */
    Ema_Model_Write(model_id, file_name);



}


/*************************************************************************/
/*
        Local Function: op_stat_init_write_ema(stat_id, string1, value1,
                                                string2, value2)

        Parameters:
                stat_id : type integer indicates the statistic to be written to
                size    : number of elements to allocate.
                string1 : label for the abscissa type char name[256]
                string2 : label for the ordinate type char name[256]

        Use: Used to create a statistic graph and write the initial values
             for the statistic. Should be used once for each statistic.

        IMPORTANT:Should be used once and only once for each statistic.
        That is, this routine must not be called twice with the same stat_id.

*/

void op_stat_init_write_ema(stat_id, size, string1, string2)

   int    stat_id;
   int    size;
   char*  string1;
   char*  string2;

{
```

```
    /* check that stat_id not to large */
    if (stat_id >= MAX_STATS) {
        fprintf(stderr,"op_stat_init_write_ema: stat id too large, value given %i\n",sta
t_id);
        exit (-1);
    }

    /*  create the stat object */
    statinfo[stat_id].stat_obj=Ema_Object_Create(model_id, OBJ_OV_VECTOR);
    statinfo[stat_id].size =size;
    statinfo[stat_id].index=-1;

    /* sets the size of the vector and store it */
    Ema_Object_Attr_Set(model_id,statinfo[stat_id].stat_obj,
                        "abscissa vec", COMP_DVEC_SIZE, size,
                        "ordinate vec", COMP_DVEC_SIZE, size,
                        EMAC_EOL);

   /* set the attributes of the vector names */
   Ema_Object_Attr_Set(model_id, statinfo[stat_id].stat_obj,
        "abscissa name", COMP_CONTENTS, string1,
        "ordinate name", COMP_CONTENTS, string2,
         EMAC_EOL);

}


/*****************************************************************************/
/*
        Function: op_stat_write_ema(stat_id, value1, value2)

        Parameters:
                stat_id : integer indicates the statistic to be written to
                value1  : value for abscissa double
                value2  : value for ordinate double

        Use: Adds the new value for the statistic to the stat vector.

        IMPORTANT: never try to increment a statistic that has not been
        created with op_stat_init_write().

*/

void op_stat_write_ema(stat_id, value1, value2)

   int stat_id;
   double value1;
   double value2;

{

    /* increment the vector index of the statistic object */
    statinfo[stat_id].index++;

    /* check that we are not exceeding limit */
    if (statinfo[stat_id].index == statinfo[stat_id].size) {
        fprintf(stderr, "op_stat_write_ema: vector length exceeded on statistic %i\n",st
at_id);
        exit (-1);
    }

/* used for debugging
    fprintf(stderr,"stat_id %d  value1 %f value2 %f\n",stat_id, value1,value2);
*/

    /* set the attributes of the vectors */
    Ema_Object_Attr_Set(model_id,statinfo[stat_id].stat_obj,
        "abscissa vec", COMP_DVEC_CONTENTS(statinfo[stat_id].index), value1,
        "ordinate vec", COMP_DVEC_CONTENTS(statinfo[stat_id].index), value2,
         EMAC_EOL);

}
```

```
void  begin_time()
{

  printf("****************************************\n   Simu. begin time: ");
  system("date");
  printf("\n*************************************\n");
 }

void  end_time()
 {
  printf("****************************************\n   Simu. end time: ");
  system("date");
  printf("\n*****************************************\n");


 }
```

```
#include  "m4.h"


typedef struct
          {
             double x;
             double y;
             double z;
          }   SAT_TYPE;

typedef struct
         {
            double X;
            double Y;
            double Z;
         } CORD_TYPE;

SAT_TYPE    sat[TOTAL_SATS],line[2],vector[4];

double      SUN_angle,MOON_angle;
CORD_TYPE   SUN, MOON;

                        /* iterative variables of grid */
double      SUN_PHI,SUN_THETA, MOON_PHI, MOON_THETA;


double      sum_affected_sats, sum_affected_links;
int         max_affected_sats, min_affected_sats;
int         current_aff_sat;
int         current_aff_link;
int         max_affected_links;
int         temp_affected_sats;
enum        flag { yes, no };

struct  {
            enum flag   confirm ;
            double      x,y,z;
            int         company;
            double      angle ;
        }     affected_sat[TOTAL_SATS];

int   grid_size, current_links_assign;

struct node
        {
          double       key;
          struct node  *left, *right;
        }  *head ,*result ;

struct sorting_array_type
  {
    int     link_sat;
    double  distance;
  } sorting_array[TOTAL_SATS];

extern  FILE  *g;



/*****************************************************************/
/* convert_to_xyz : convert longitude, latitude, and altitude      */
/*                  to x, y and z coordinates */
```

```
/*****************************************************************/
int convert_to_xyz(index)
int index;
{
     double  longitude, latitude, altitude,distance;
     int     i;

     for(i=0;i<TOTAL_SATS; i++)
       {
         latitude  = SatLat(i,index);
         longitude = SatLong(i,index);
         altitude  = SatAlt(i,index);
         distance =  EARTH_RADIUS + altitude;

         sat[i].y = distance*cos(latitude*M_PI/180.0)*
                        cos(longitude*M_PI/180.0);
         sat[i].x = distance*cos(latitude*M_PI/180.0)*
                        sin(longitude*M_PI/180.0);
         sat[i].z = (-1)*distance *sin(latitude*M_PI/180.0);



         SatX(i,index) = sat[i].x;
         SatY(i,index) = sat[i].y;
         SatZ(i,index) = sat[i].z;

       } /* end of for loop */
} /* end of convert_to_xyz */


/*-------------------------------------------------------------------*/
/* compare() : The compare fun used at qsort().                      */
/*-------------------------------------------------------------------*/

int compare(x,y)
  struct sorting_array_type *x;
  struct sorting_array_type *y;
 {
      return (x->distance - y->distance );
 }

/*-------------------------------------------------------------------*/
/* link_assign(): assign links in table .                            */
/*-------------------------------------------------------------------*/
int  link_assign(index)

  int index;
  {
   int  i,j,k;
   double  a[10];
   double distances[TOTAL_SATS];
   double dist_x,dist_y,dist_z;

   for(i=0;i<TOTAL_SATS;i++)
    {
       for(k=0;k<TOTAL_SATS;k++)
        {
                                  /* in order to sort array */
          sorting_array[k].distance=11111111111.;
          sorting_array[k].link_sat=k;       /* original array index */
        }
    for(j=i+1;j<TOTAL_SATS;j++)
     {
       dist_x = SatX(i,index)-SatX(j,index);
```

```
      dist_y = SatY(i,index)-SatY(j,index);
      dist_z = SatZ(i,index)-SatZ(j,index);
      sorting_array[j].distance = sqrt(dist_x * dist_x +
                                  dist_y * dist_y + dist_z * dist_z);
    }

  qsort(sorting_array,TOTAL_SATS,sizeof(struct sorting_array_type), compare);


 /* find the topest 6 elements for every line element */

  for(k=0;k<MAX_LINKS;k++)
   if(sorting_array[k].distance <= MAX_DIST)
     {
      current_links_assign++;
      table[i][sorting_array[k].link_sat] = 1;
     }

 }

printf("the total links assigned are %d\n",current_links_assign);

} /* end of sort_assign */


/*------------------------------------------------------------------------ */
/* comput_angle : compute the angle between two line.                      */
/* ------------------------------------------------------------------------ */

double  comput_angle(line_1, line_2, index)
        SAT_TYPE  line_1;
        SAT_TYPE  line_2;
        int index;
{
 double    product, len_line1, len_line2;
 double    temp, deg_rad,temp1, temp2;
 double     deg;

      /* Dot product of two lines */
      product = line_1.x * line_2.x + line_1.y * line_2.y +
                line_1.z * line_2.z ;
      /* Get square  length of the first line */
      temp = line_1.x * line_1.x + line_1.y * line_1.y +
                line_1.z * line_1.z;
      /* Get length */
      len_line1 = sqrt(temp);

      /* Get square length of second line */
      temp1 = line_2.x * line_2.x + line_2.y * line_2.y +
                line_2.z * line_2.z;
      /* Get legth */
      len_line2 = sqrt(temp1);


     if (( len_line1 * len_line2 == 0) || (product == 0) )
        {

       fprintf(g,"\n--- SUN_PHI=%f.1, SUN_THETA=%f.1 ----",SUN_PHI,SUN_THETA);

          fprintf(g,"\nlline_1.x=%f,\nline_1.y=%f,\nline_1.z=%f",
                   line_1.x,line_1.y,line_1.z);

          fprintf(g,"\nline_2.x=%f,\nline_2.y=%f,\nline_2.z=%f",
                   line_2.x,line_2.y,line_2.z);

          fprintf(g,"\n\nlen_line1=%f,\nlen_line2=%f",len_line1,len_line2);
          fprintf(g,"\nproduct=%f",product);
          fprintf(g,"\ntemp2 = %f",temp2);
          if ( fflush(g) == EOF)
            printf("\nCan't do fflush ");
        }


      temp2 = product/(len_line1 * len_line2);

     if (( len_line1 * len_line2 == 0) || (product == 0) )
        {
          fprintf(g,"\ntemp2 = %f",temp2);
        }

                                /* get angle expressed in radian */
      deg_rad = acos( temp2 );

      deg =  deg_rad * 180 /M_PI ;   /* get angle expressed in degree */

       return deg;
   }  /* END OF  angle_comput */


/* ------------------------------------------------------------------------  */
/* update_affected_sats: The procedure uses distance to choose one sat to */
/* become original point of vector, then call comput_angle to get angle.  */
/* Use angle to update those variables.                                    */
/* ------------------------------------------------------------------------ */

update_affected_sats(index)
int index;

{
int     *near_sat , *far_sat ;
int     i,j;
double  len_line[2];



SUN.X = SUN_R * sin(SUN_THETA * M_PI/180.0) * cos(SUN_PHI * M_PI/180.0);
SUN.Y = SUN_R * sin(SUN_THETA * M_PI/180.0) * sin(SUN_PHI * M_PI/180.0);
SUN.Z = SUN_R * cos(SUN_THETA * M_PI/180.0);

   /* Initialize to 0 for every sat. */
for(i=0;i<TOTAL_SATS;i++)
   affected_sat[i].confirm = no;

 /* Below decide which sats is further away from SUN or MOON , so later we can correct. W
e use PRODUCT formula  */

for(i=0; i<TOTAL_SATS ; i++)
 {
 for(j=i+1; j<TOTAL_SATS  ;j++)
   {
    if(table[i][j]== 1)
     {
       /* compute SUN vector */
       line[0].x = SUN.X - sat[i].x;
       line[0].y = SUN.Y - sat[i].y;
       line[0].z = SUN.Z - sat[i].z;

       line[1].x = SUN.X - sat[j].x;
```

```
        line[1].y = SUN.Y - sat[j].y;
        line[1].z = SUN.Z - sat[j].z;

        /* length of line[0] */
        len_line[0] = line[0].x * line[0].x + line[0].y * line[0].y +
                      line[0].z * line[0].z;

        /* length of line[1] */
        len_line[1] = line[1].x * line[1].x + line[1].y * line[1].y +
                      line[1].z * line[1].z;

        /* decide which sat is further away form SUN */
        if (len_line[0] > len_line[1])
           {
            far_sat  = &i;
            near_sat = &j;
            }
        else
           {
             far_sat  = &j ;
             near_sat = &i ;
           }

        /* Generate two vectors associated with SUN.*/
        vector[0].x = sat[*near_sat].x - sat[*far_sat].x ;
        vector[0].y = sat[*near_sat].y - sat[*far_sat].y ;
        vector[0].z = sat[*near_sat].z - sat[*far_sat].z ;
        vector[1].x = SUN.X - sat[*far_sat].x;
        vector[1].y = SUN.Y - sat[*far_sat].y;
        vector[1].z = SUN.Z - sat[*far_sat].z;

       /* Get angle between two vectors */
       SUN_angle   = comput_angle(vector[0],vector[1],index) ;


      /* update value for  each sat.*/

      if (SUN_angle <  max_view_for_sun )
       {
        if ((affected_sat[i].confirm == no)&&(affected_sat[j].confirm == no))
                        current_aff_sat += 2;

        if ((affected_sat[i].confirm == yes)&&(affected_sat[j].confirm == no))
                        current_aff_sat += 1;

        if ((affected_sat[i].confirm == no)&&(affected_sat[j].confirm == yes))
                        current_aff_sat += 1;

        current_aff_link++;

        affected_sat[i].confirm = yes;
        affected_sat[j].confirm = yes;

       }  /* end of if  */
     }
   }
 }


} /*   END OF get_angle */


/*-----------------------------------------------------------------------*/
/* link1(): Every time this procedure is called. It will executed link2()     */
/*          for different SUN and MOON positions, and will get the number of  */
/*          affected sats. This  is stored in an element of an array          */
/*          and later print out.                                              */
/*-----------------------------------------------------------------------*/

link1(index)
int  index ;

{
 int  reusltnode= -1 ;
 int  time;

 /* initilize value */
 max_affected_sats = 0;
 min_affected_sats = TOTAL_SATS;
 sum_affected_sats = 0;
 sum_affected_links = 0;
 max_affected_links = 0;
 grid_size=0;
 current_links_assign=0;


 /* Only update links assignments by inc_index value to get efficient */
 if(index%times_of_index_step == 0)
  {
      convert_to_xyz(index);
      link_assign(index);
      fprintf(stderr,"\nthe index = %d  ", index);
   }

    fprintf(g,"\n******** index = %d  *********** \n ", index);



 for(SUN_PHI =161;SUN_PHI< (161+GRID_SUN_PHI); SUN_PHI += PHI_INC)
  {
    for(SUN_THETA= 87;SUN_THETA< (87+GRID_SUN_THETA); SUN_THETA += THETA_INC )
     {
       current_aff_link = 0;
       current_aff_sat = 0;


       update_affected_sats(index);

       if (max_affected_links < current_aff_link)
            max_affected_links = current_aff_link;

       if (max_affected_sats < current_aff_sat)
            max_affected_sats = current_aff_sat ;

       if (min_affected_sats > current_aff_sat)
            min_affected_sats = current_aff_sat ;

       sum_affected_links += current_aff_link;
       sum_affected_sats += current_aff_sat ;
       grid_size++;


     }
  }


 time= time_step *index ;
```

```
/* output  */
op_stat_write_ema(1,(double)time,(double)max_affected_sats);
op_stat_write_ema(2,(double) time,(double)min_affected_sats);
op_stat_write_ema(3,(double)time,(double)sum_affected_sats/grid_size);

op_stat_write_ema(4,(double)time,(double)max_affected_links);
op_stat_write_ema(5,(double)time, (double)min_affected_sats/2.0);
op_stat_write_ema(6,(double)time,(double)sum_affected_links/grid_size);

op_stat_write_ema(7,(double)time,
                    ((double)max_affected_sats)/TOTAL_SATS);
op_stat_write_ema(8,(double)time,
                    ((double)max_affected_links)/current_links_assign);

fprintf(stderr,"\nmax_affected_sats=%d",max_affected_sats);
fprintf(stderr,"\nmax_affected_links=%d\n",max_affected_links);

}  /* end of link1() */
```

```
The incline = 27.500000
The sorted lon_ascen are  0.0  3.0  6.0  8.0  9.0 11.0 18.0 20.0 23.0 24.0 29.0 34.0 40.0
46.0 50.0 52.0 60.0 61.0 73.0 81.0 85.0 89.0 92.0 99.0 103.0 107.0 118.0 120.0 126.0 132.0
 146.0 148.0 150.0 160.0 165.0 166.0 172.0 173.0 179.0 184.0 200.0 203.0 213.0 224.0 227.0
 240.0 241.0 252.0 256.0 261.0 262.0 269.0 274.0 275.0 281.0 282.0 284.0 288.0 290.0 292.0
 295.0 296.0 298.0 299.0 301.0 305.0 306.0 310.0 311.0 315.0 317.0 329.0 331.0 335.0 337.0
 339.0 344.0 349.0 354.0 355.0
-----------------------------------------

The incline = 57.500000
The sorted lon_ascen are  0.0  3.0  7.0  8.0 15.0 21.0 27.0 29.0 30.0 32.0 38.0 40.0 41.0
42.0 43.0 47.0 56.0 59.0 61.0 63.0 65.0 67.0 68.0 71.0 84.0 89.0 92.0 94.0 97.0 107.0 118.
0 129.0 137.0 142.0 145.0 154.0 158.0 159.0 161.0 166.0 167.0 168.0 170.0 175.0 181.0 182.
0 197.0 198.0 201.0 204.0 211.0 217.0 218.0 221.0 225.0 226.0 232.0 240.0 241.0 243.0 245.
0 259.0 260.0 261.0 282.0 297.0 298.0 301.0 304.0 305.0 307.0 309.0 315.0 318.0 321.0 337.
0 341.0 344.0 346.0 349.0
-----------------------------------------


The incline = 90.000000
The sorted lon_ascen are  1.0  8.0 10.0 12.0 18.0 29.0 33.0 41.0 43.0 45.0 47.0 51.0 53.0
67.0 70.0 72.0 76.0 80.0 96.0 99.0 101.0 102.0 111.0 117.0 119.0 122.0 132.0 133.0 134.0 1
37.0 154.0 162.0 176.0 177.0 178.0 181.0 185.0 194.0 200.0 202.0 203.0 205.0 207.0 213.0 2
14.0 215.0 218.0 223.0 224.0 226.0 227.0 228.0 229.0 230.0 232.0 233.0 241.0 245.0 249.0 2
53.0 254.0 257.0 268.0 270.0 287.0 292.0 295.0 301.0 308.0 312.0 313.0 315.0 318.0 320.0 3
23.0 326.0 327.0 329.0 347.0 359.0
-----------------------------------------

*********** End of generating orbit ************
```