Technical Report 2007-05-01
Internetworking and Media Communications Research Laboratories,
Department of Computer Science, Kent State University
http://medianet.kent.edu/technicalreports.html

Technical Report: TR2007-05-01

An Implementation Road Map: Space Open Shortest Path First Routing Protocol

Thomas K. Hamilton

Prepared for Prof. Javed I. Khan

Internetworking and Media Communications Research Laboratories,
Department of Computer Science,
Kent State University, Kent OH 44242
thamilto@cs.kent.edu

May 2007

Abstract: *Soon there will be a need for a space protocol robust enough to provide reliable network support in space. Although a protocol has been defined and proved to be effective, many steps remain before this protocol can be implemented and deployed successfully. This report plots an initial general plan for the SOSPF protocol implementation and identifies gaps in the protocol that could potentially be areas for future research in the field of space routing.*

1. SOSPF Protocol

SOSPF, or Space Open Shortest Path First, is a routing protocol specification meant to provide a satisfactory solution to the problems inherent in space routing. The topology of a space network is extremely volatile as both the end-points and routers themselves are constantly moving. This sort of situation is known as an ad-hoc network topology and is one of the main problems that the SOSPF protocol must address. It does so by using a spacial predictability model that takes advantage of the fact that any distinct celestial object is either bound by an orbit (planets/satellites) or by a known trajectory and speed (spacecraft); in other words, the positions of these objects are governed by known mathematical formulas. Using these formulas in conjunction with six orbital parameters specific to a celestial object at any given time, the exact position of that object can be calculated. The predictable mobility model of space routing is at the core of the SOSPF protocol and it spends much of its time keeping track of when and for how long it will be able to communicate with its neighboring routers. SOSPF is the first known space routing protocol specification that takes advantage of the naturally hierarchical nature of space in order to promote information hiding between logical network areas in order to reduce the overhead of SOSPF protocol traffic; thus, a well-designed, segmented SOSPF network that takes advantage of this hierarchical organization has the potential for higher percentage of bandwidth available for data transfer than if it did not. SOSPF also supports routing when intermittent links exist in the network which is relatively uncommon for TCP/IP network but unavoidable in a space routing environment. Intermittent paths are dealt with by storing data in transit along its route and forwarding it when the next hop becomes available. This process causes additional protocol overhead but will be a necessity until enough routers exist in space in order to invalidate the need for intermittent path resolution functionality. Through the implementation of the predictable space mobility principle and intermittent link handling, an SOSPF network designed with the space hierarchy in mind will provide an environment with low delay, high throughput with a relatively fast convergence after a network topology change. SOSPF promises to become a complete space routing solution in the near future.

2. SOSPF Protocol Implementation Overview

In the research on which this implementation proposal is based on, it was decided that the SOSPF protocol would not have to be built from the ground up; rather, the existing OSPFv3 (IPv6) code base could be used as a starting point for the work. This is because the existing OSPF implementation already offers much of the functionality that SOSPF requires.
The features of OSPF that make it suitable as the core of SOSPF are:

1. OSPF allows the routing domain to be divided into multiple logical areas within one autonomous system (AS). This sort of division allows for the hierarchical organization discussed earlier thus providing a routing environment conducive to information hiding. This leads to less protocol overhead traffic which is supremely important in a space environment where links may be low bandwidth with a high propagation delay.
2. OSPF maintains a consistent view of the routing domain within its area by synchronizing the constituent routers' Link State Databases (LSDs). LSD consistency through a logical area is crucial to the efficient operation of SOSPF as it may not be able to make suitable routing decisions otherwise.
3. OSPF provides a connection-oriented environment by way of it's “Hello” sub-protocol. SOSPF's inherently volatile topology requires such a way to keep track of which routers are still reachable and which are not.

4. OSPF has been proven to be able to detect changes in network topology in a finite time. The ability to

5. do this is very important to the efficient operation of the proposed SOSPF protocol; in fact, topological changes should be detected and dealt with in as fast a time as possible as EVERY end point and router is continuously in motion in a space environment.
6. OSPF uses master/slave semantics in order to reduce protocol overhead traffic on links. As mentioned earlier, reducing any unnecessary traffic on links in space is of utmost important as links tend to be of lower bandwidth with higher propagation delays than their terrestrial counterparts.
7. OSPF implements a flooding scope semantic which allows for a loop-free routing environment. Loop-free environments work to reduce protocol traffic overhead. It has already been discussed why reducing traffic overhead is important for SOSPF.

Although OSPF provides much of the functionality that SOSPF requires, there is still much to be done in order to successfully implement the space implementation of OSPF.

Terrestrial OSPF is **NOT** suitable for space routing for the following reasons:

1. The area division mechanism that OSPF provides is insufficient for space routing. SOSPF requires two different “types” of areas (Satellite Constellation Area and Celestial Object Area) that behave differently whereas OSPF just provides a generic idea of area with no way to distinguish between different area types.
2. In a colony based (Earth) OSPF network, routers are in fixed positions. Routes between these routers are calculated using Djikstra's shortest path algorithm. SOSPF must account for the fact that all the routers in the network are constantly on the move; otherwise, inaccurate routes could be calculated.
3. SOSPF requires support for intermittent links due to router occlusion due to celestial objects both to avoid unnecessary LSD synchronization when contact is reestablished, but also to ensure any Link State Advertisements (LSAs) received during the occlusion are exchanged between the neighbors.

The three points discussed above may make it sound like implementing SOSPF will be easy; however, this is far from the case. After analyzing the existing OSPF code base and the SOSPF requirements, I have identified the following list of things that will need to be done in order to implement SOSPF successfully (Each item will be discussed in detail in section IV of this proposal):

1. Expand robustness of logical areas to allow differentiation between logical area types, namely Satellite Constellation Areas and Celestial Object Areas.
2. The role of Area border routers must be modified to include the notions of Satellite Constellation Border Routers and Celestial Object Border Routers (SCBRs, COBRs). Routers must have the ability to belong to multiple areas simultaneously (new router level configuration parameter) in some cases and must allow routers to be excluded from being sent out in area summaries (DoNotAdvertise bit).
3. The neighbor data structure must be modified to include numerous SOSPF specific parameters.
4. A new neighbor state, “Sleeping” will be added. The neighbor state transition logic will also need to be modified in order to account for this new state. Logic surrounding the transitions to and from the Sleeping state will need to be implemented as well.
5. A new Area-Level data structure known as the Neighboring Routers List will need to be defined and integrated with existing code. This data structure will provide a way for an area to know which neighboring routers are within its area at different times.
6. A Space Predictable Model for Space Predictable Mobility sub-system will need to be defined and implemented in order to provide each router the ability to calculate the exact position of any other router it neighbors at any given time.
7. Two new Link State Advertisement types will be added to support the transfer of data specific to the

8. SOSPF routing protocol throughout the routing domain. These two LSAs are the Space Router LSA (SR-LSA) and Area Membership LSA (AM-LSA). SR-LSAs provide information regarding propagation delays on links at specific times whereas AS-LSAs notify other routers of when and for how long the sending router will be a member of a specific area or areas. The logic used to determine when to generate and distribute these LSAs will also need to be implemented.
9. Three new router-level data structures will need to be added. These data structures are the Area-Membership list , the Area Border Router List and the Space Routers list. These data structure will keep track of which areas a router belongs to at any given time (among other things). Logic for initial generation and maintenance of these data structures will be required.
10. The Flooding Scope semantics will need to be modified to provide the two levels of flooding functionality required by SOSPF. These two flooding scopes are Internal Flooding Scope and Celestial Flooding Scope.
11. Multiple changes will be made to the "Hello" sub-protocol. Both the actual structure of each individual "Hello" packet will need modification as well the logic needed to establish and sever bidirectional relationships between neighboring routers.
12. The "Exchange" sub-protocol will required modification as well in order to ensure correct behavior upon receipt of one of the new LSA types (SR-LSA, AM-LSA).
13. The SDIP routing algorithm described in the research will required implementation and integrationwith the existing OSPF code base in order for SOSPF to be successfully implemented. SDIP is an effective algorithm for calculating the approximate optimum path in an environment containing intermittent links. This algorithm will replace Djikstra's routing algorithm in SOSPF.Before delving any deeper into what will be required for each of these 12 action items, it will be wise to become acquainted with the existing OSPF code base and to identify files that will probably required modification in order to achieve our goal of SOSPF implementation.

3. OSPF Code Base Analysis & Discussion

Note: Due to trouble finding the source code for OSPFv3, the source for OSPFv2 was used for reference. The only notable difference between the two versions is that v2 uses IPv4 whereas v3 uses IPv6. SOSPF is based on the IPv6 addressing scheme so if a suitable implementation of OSPFv3 is unable to be found at the time of implementation, it will also be necessary to modify the IP address semantics throughout the OSPF code base to use IPv6. Discussion of this conversion will not be discussed further in this preliminary implementation plan.

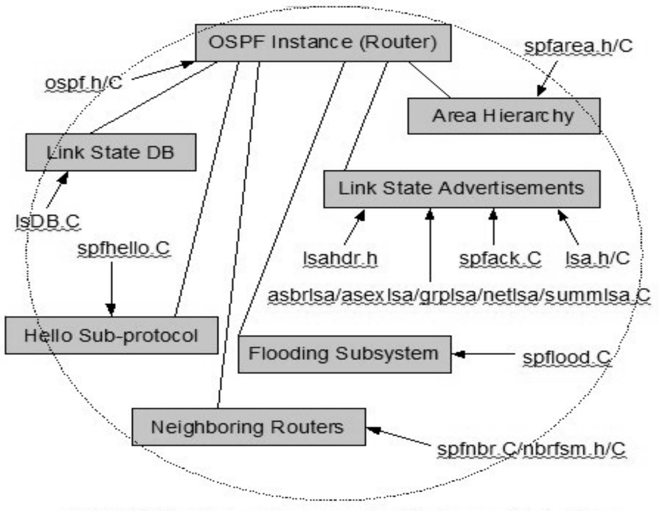
What follows is a list of each file in the OSPF code base that will likely need to be changed in the process of implementing SOSPF. Each listing will include the name(s) of the files and what its purpose is.

1. lsdb.C: This file contains the routines that manipulate the OSPF link-state database. The database itself is represented by the LSdb class which is implemented as an AVL tree.
2. lsa.h/lsa.C: Defines a generic representation of an LSA when stored in the router's database. Each LSA is like a node in the AVL tree of the LSdb.
3. ospf.h/ospf.C: contains OSPF base class. contains all data necessary to run a single instance of the OSPF protocol.
4. ip.h: necessary definitions for the IP protocol
5. ospfinc.h: A central list of OSPF include files.
6. spfarea.h: Defines the OSPF area class. One defined for each OSPF area.
7. spfarea.C: Defines routines implementing OSPF area support.
8. spfhello.C: Implements the routines required for sending and receiving of Hello packets.
9. spfack.C: Routines implementing the receiving and sending of Link State Acknowledgment packets.
10. spfcalc.C: Routines implementing the OSPF routing calculation. This includes the main routine called to rebuild the routing table, the Dijkstra calculation, routine for scheduling appropriate routing calcs, and

routines to detect changes in the routing table and take appropriate actions.

11. spflood.C: Implementation of the OSPF flooding algorithm.
12. spfnbr.C: deals with basic OSPF neighbor operations (most OSPF neighbor implementation are in nbrfsm.h).
13. spfpkt.h: OSPF packet structure definitions.
14. spfutil.C/spfutil.h: Utility functions and definitions for packets and FSM transitions.
15. asbrlsa.C: Routines dealing with area border router Summary-LSAs. originating, parsing, unparsingand running routing calculations.
16. asexlsa.C: Routines implementing AS-external-LSAs including originating, parsing and using AS-external-LSAs in routing calculations.
17. grplsa.C: dealing with Group-Membership-LSAs. includes originating, parsing and running routing calculations.
18. lsahdr.h: defines common 20-byte header for all LSAs.
19. nbrfsm.h/nbrfsm.C: possible OSPF neighbor states. (Neighbor FSM? What is FSM?)
20. netlsa.C: Originate a network-LSA (ordinary LSA).
21. rte.h/rte.C: definitions of routing table and its entries as well as routines for manipulating it.
22. rtlsa.C: Routines dealing with the origination, parsing and unparsing of router-LSAs.
23. summlsa.C: routines dealing with Summary-LSAs. includes origination, parsing, unparsing and running routing calculations.

The following diagram shows the main components of OSPF and what source code files hold the implementation of that particular part of the system.



OSPF Main System Components with Source Code Files

The SOSPF implementation will also involve adding additional files. It would be prudent and recommended to adhere to the standards apparent in the existing OSPF code base when adding these new files. The new files to be added will be discussed in the next section in the order they are encountered.

4. Implementation Details

1. Increase robustness of area hierarchy. Allow for different logical area types; specifically, Satellite Constellation Areas and Celestial Object Areas.
File(s)s affected:

spfarea.h
spfarea.C

Change Description:

An SOSPF network is composed of one or more areas. Each area in the network is either a Satellite Constellation Area or a Celestial Object Area. The area object will need to be modified to keep track of this information. The area object will also need to have objects added to keep track of the two new types of LSAs to be implemented for SOSPF.
Modify Data Structure SOSPF SpfArea class structure (spfarea.h):

Add constants:

```
#define CELESTIAL_OBJECT_AREA 0 /* COA */
#define SATELLITE_CONS_AREA 1 /* SCA */
```

Add fields:

```
private int area_type; /* COA or SCA */
AVLtree srLSAs; /* Space-Router LSAs */
AVLtree amLSAs; /* Area-Membership LSAs */
```

Modify SpfArea Constructor (spfarea.C):

Simply initialize the area_type to 0. /* COA */

Add friend functions (spfarea.h/spfarea.C):

```
void *OSPF::SetAreaType(const SpfArea* area, int type);
int *OSPF::GetAreaType(const SpfArea* area);
```

The above changes should be sufficient to allow other parts of the protocol to determine what type of area a given SpfArea object is as well as provide storage locations (AVLtrees) for the newly implemented SOSPF-specific LSA types.

2. An area border router must also have a type distinction added in order to correctly fulfill their role. Other area border router functionality will remain unchanged. The two types of area border routers are Satellite Constellation Border Router (SCBR) and Celestial Object Border Router (COBR). Note that a COBR is also an SCBR.

Affected files:

ospf.h
ospf.C

Change description:

The ospf.h and ospf.C files will need to be modified to allow the ability to distinguish whether or not the router is a SCBR or a COBR. At first I was looking for some sort of Area Border Router class however, once I realized that since only a single instance of the OSPF protocol (or in this case SOSPF) is running on a

particular router, that router level parameters were most likely stored in the OSPF data structure. On a side note, when the actual implementation is done, it may be wise (or maybe not wise) to rename all instances of ospf in the code to sospf to more accurately reflect the actual protocol. This might cause a lot of pain in testing though since almost everything is referencing these structures. For this change however, we will simply be adding a single variable to the OSPF structure in order to allow it to tell what type of router it is. There should be three values, 0 for normal router, 1 for SCBR and 2 for COBR. The value should be initialized to 0 for all new instances. Accessor and mutator functions should also be provided as friend functions to allow modification. While we're modifying this structure, we also want to add additional flags for DoNotAdvertise and MultiAreaMem.
Modify data structure OSPF (ospf.h):

Add constants:

```
#define ROUT 0
#define SCBR 1
#define COBR 2
```

Add fields:

```
int router_type; /* ROUT, SCBR, COBR */
int DoNotAdvertise; /* router existence propagated outside area?*/
int MultiArea; /* Can this router belong to multiple areas? */
```

Modify OSPF structure constructor (ospf.C):

Initialize router type to ROUT, DoNotAdvertise to 0 and MultiArea to 0.
Add friend functions (ospf.h/ospf.C):

```
void OSPF::SetOSPFRouterType(int type);
int OSPF::GetOSPFRouterType();
void OSPF::SetOSPFDoNotAdvertiseBit(int value);
int OSPF::GetOSPFDoNotAdvertiseBit(();
void OSPF::SetOSPFMultiAreaBit(int value);
int OSPF::GetOSPFMultiAreaBit(();
```

These changes should allow an area border router to take the appropriate action as determined by its type as well as provide the flags that will provide a way for a router to hide itself to areas external from its own as well as control whether or not it can be part of multiple areas simultaneously.

3. Some additional SOSPF specific router-level parameters must be added as well as some to the the neighbor data structure. Parameter to add to the neighbor data structure include maximum stability period and a propagation list. Each entry in the propagation list should contain birth time, propagation period, and propagation delay. Also to be added are a New LSAs list (LSA headers received while sleeping) and an LSAs threshold (max number of LSAs to store in New LSAs list). Router level additions include a Area Membership List, an Area Border Routers List and a Space Routers List.
Affected Files:
ospf.h
ospf.C
spfnbr.h
spfnbr.C

Change Description:

Modify the neighbor data structure to keep track of propagation delay between the two neighbors at different times. Additionally, allow each neighbor data structure to keep track of all LSAs received while in the sleeping state. Constrain this list of received LSAs with a threshold variable.
New File: proplist.h (define a PropagationListEntry class and add constants)

```
#define MAX_STABILITY_PERIOD ? /* not sure what a value would be */

class PropagationListEntry
{
  double birthtime; /* in ms */
  double propagation_period;
  double propagation_delay; };
/* I guess we don't really need a specific class to hold these... could
just use a generic list or vector */
```

Modify data structure SpfNbr (spfnbr.h):

Add fields:

```
int LSAThreshold;
AVLTree newLSAs;
```

Modify data structure OSPF (ospf.h):

```
SpfArea areaMembershipList[];
rtid_t ABRList[]; /* list of router Ids */
rtid_t SpaceRoutersList[]; /* list of router Ids */
```

4. Add a new neighbor state called Sleeping. This will be the state that a specific SpfNbr structure will go to when its constituent routers become occluded by a celestial body. Logic must also be implemented in order to store received LSAs while in the Sleeping state into the SpfNbr's newLSA list (defined earlier). Additionally, logic will have to be defined for the transition out of Sleeping and then to flood all LSAs in the NewLSAs list with the appropriate flooding scope.
Affected Files: nbrfsm.h, nbrfsm.C, spfnbr.h, spfnbr.C, ospf.h, ospf.C

Add Sleeping neighbor state (nbrfsm.h):

Add the following value to the first enum in the file:

```
NBS_SLEEP = 0x90;
```

Add OSPF neighbor events for controlling transitions to and from the Sleeping state.

Full -> Sleeping (Sleep)
Sleeping -> Down (Reached Max, No Recurrence)
Sleeping -> Exchange (Awaken and Unsynchronized)
Sleeping -> Full (Awaken and Ready)

Modify data structure FsmTran (nbrfsm.C):

Modify the FsmTran datastructure to include the four new transitions defined above.

Not entirely sure how to do this correctly...
Modify the char *nbrstates(int state) function to include the NBS_SLEEP state.

Continue in this manner through the file, adding the NBS_SLEEP state and its associated transitions. You will also have to add event transition logic. Transitions to NBS_SLEEP state will have to somehow indicate that any received SR or AM LSAs while in the NBS_SLEEP state will need to be stored in the NewLSAs list. Then, transitions FROM the NBS_SLEEP state will need logic to flood all LSAs in the NewLSAs list (if any) using the appropriate scope.

By modifying the neighbor state transition model for SOSPF, we have given the protocol the ability to deal with router occlusion, an inevitability in a space environment.

5. I think the NeighboringRoutersList data structure already exists in normal OSPF so I'll need to look into this to see if anything additional actually has to be implemented.

6. In order for a router to be able to predict the movement of each of its neighbors, the neighbor data structure maintained for each router neighboring the one on which the software is running will need to keep track of the six orbital parameters for the neighbor including the calculating method in order to be able to calculate its location at any given point in time.
Affected Files:
spfnbr.h
spfnbr.C
ospf.h
ospf.C

Change Description:
Add the six orbital parameters to the data structure as well as a calculating method value. Because I have no working knowledge of how trajectories and orbit locations are calculated in space. The calculating method variable should be an object of a new type calculating method. All implemented calculating methods should inherit from this base object and will be required to override a single abstract method known as calculate_location_at_time. This will return an object of another new class called 3dloc. It will contain the three-dimensional location of the object in space at the specified location. It will take the six orbital parameter values as well as the time you wish to calculate the location at. These changes are necessary in order to implement the Predictable Space Mobility principle on which SOSPF depends on.
Create new file: 3dloc.h/3dloc.C
Create new class:

```
class 3DLoc /* I have no idea if this is actually how they would represent
a celestial object location */
{
  int x;
  int y;
  int z;
};
```

Add 3DLoc.h to ospfinclude.h.

Create new file: calc_method.h
Create new class:

```
class CalcMethod
```

```
{
  3DLoc calc_loc(char center_foci_axis, double ecc, double long_asc_r,
double perigree, double inc_orbit, double true_angle, double time);
};
```

Add calc_method.h to ospfinclude.h.

Modify SpfNbr data structure (spfnbr.h):
Add fields:

```
char center_foci_axis;
double eccentricity;
double longitute_asc_sospf_rout;
double arg_perigree;
double inc_of_orbit;
double true_angle;
CalcMethod calculating_method;
```

Modify OSPF data structure (ospf.h):

Add fields:

```
char center_foci_axis;
double eccentricity;
double longitute_asc_sospf_rout;
double arg_perigree;
double inc_of_orbit;
double true_angle;
CalcMethod calculating_method;
```

These changes will allow a router to calculate the exact location of any of its neighbors at any given time. A timer will also need to be implemented in order to update a router's own orbital parameters.

7. Add two new LSA types: Space Router LSAs and Area-Membership LSAs. The SR-LSA will be used to describe the quality of the connection between a pair of SOSPF routers by calculating the estimated propagation delay between them during a maximum stability period (defined earlier). The AM-LSA will advertise which areas a particular SOSPF router belongs to and for how long it will belong to each of those areas. The new LSA types shall be implemented using the same structure as the existing LSA types in OSPF.
Add new files: srlsa.C, amlsa.C
Modify file: lsa.h
Define srLSA and amLSA classes that inherit from the generic LSA class.
Add the following variables to the srLSA class:

```
ipaddr_t source; /* source router IPv6 address */
ipaddr_t dest; /* destination router IPv6 address */
```

A list of tuples with the following data (you may want to define a class for this particular tuple type):

```
int begintime; /* when dest comes in view of source */
int connection_period; /* how long in view */
int propagation_delay; /* delay incurred from this connection */
```

Add the following variables to the amLSA class:

```
ipaddr_t source; /* source router's IPv6 address */
int AMForwardingList[]; /* areaIDs to forward to */
```

A list of tuples with the following data (you may choose to define a class for this tuple type):

```
int areaID; /* The area ID */
SpfArea areaMembers[]; /* A list of the area members */
int areaStartTime; /* start point for area definition */
int areaPeriod; /* how long definition is valid */
/* areaMembers might be more suited to hold a list of areaIDs (ints)*/
```

Generation and flooding logic for these two types of LSAs will also need to be implemented. SR-LSAs are generated whenever an SOSPF router becomes aware of a change of any of its direct links to other SOSPF routers (page 49-50 in the research describe the six particular cases in detail). AM-LSAs are generated based on a timer or whatever the topology changes. In order to implement AM-LSA functionality, a new timer must be added as well. We shall call this the AM-Timer (Area Membership Timer). A good start to the implementation would be:

Add AMTimer (ospf.h):

```
class AMTimer : public Timer
{
    public:
    virtual void action();
};
```

Successful implementation of SR and AM LSAs will allow an SOSPF router to communicate data important to its operation to other SOSPF routers. This information will allow the routing protocol to best select routers (by having up to date estimated propagation delays) as well as maintain an accurate view of the area hierarchy which will reduce overall bandwidth usage (hopefully).

8. These data structures were implemented as part of step 4.

9. Define the two SOSPF flooding scopes: Internal Flooding Scope and Celestial Flooding Scope. These two flooding scopes are necessary in order to limit the amount of frivolous information sent across the network. After looking at the existing OSPF code, it appears that the equivalents of these two flooding scopes already exist. They are referred to as local_flood (Internal Flooding Scope) and global_flood (Celestial Flooding Scope). If any changes to flooding scope semantics are required in order to implement SOSPF, I would imagine them to involve modifying the behavior of these existing flooding scopes as opposed to defining completely new ones.

10. The "Hello" sub-protocol will require changes in order to implement SOSPF. The "Hello" packet structure will need to be modified to include SOSPF specific information. Also, the protocol will have to sever already establish bidirectional relationship due to unanswered "hello" packets (which it already does) but it must then also initiate SR-LSA flooding using the appropriate SOSPF flooding scope.

Affected Files: spfpkt.h, spfhello.C

Modify HloPkt data structure:

Add fields:

```
char center_foci_axis;
double eccentricity;
double longitute_asc_sospf_rout;
double arg_perigree;
double inc_of_orbit;
double true_angle;
CalcMethod calculating_method;
int areaID; /* maybe a list? */
```

SOSPF is a connection-oriented protocol; meaning that, all SOSPF routers that have already established a bidirectional connection will continue to exchange Hello packets. By adding this SOSPF specific data to the Hello packets themselves, we can guarantee that a pair of adjacent routers will always know, or be able to calculate the exact position of its neighbor.

11. The "Exchange" sub-protocol will require changes in order to implement SOSPF. New logic must be implemented for when an SOSPF router s receives an AM-LSA for area i that was generated by SOSPF router y and area i's entry in x's Area Membership list does not contain y in the Area Members field. It must then recalculate its Area Membership List and proceed to form bidirectional communication and neighboring relationship with y. If an LSA (other than AM-LSA) is received during the Exchange process from a router that is not currently in the Space Router's List, the router must record that router's ID in the New Space Router's List. The Space Router's List will be processed at the end of the Exchange process. A coordinate request packet is sent to the neighbor for every new SOSPF router. the rest of the coordinate request process will need to be implemented as well and is described on page 72. New logic will also have to be implemented for when an SR-LSA is received while two neighboring routers are in the Full state and the SR-LSA was generated by an SOSPF router that is not in the Space Routers List (page 72).

12. Implementation of the SDIP routing algorithm as well as integrating it with SOSPF will be a large part of the SOSPF implementation process. Implementation of this algorithm looks to be fairly straightforward. Hopefully it will be as simple as replacing any call to Djikstra's algorithm with a call to the SDIP algorithm in the SOSPF implementation.

5. Protocol Gaps

There are a number of issues that were not fully explained or tested in the research that would be suitable topics for further research. In order to implement SOSPF, these topics would need to be revisited and expanded upon.

1. Calculating Method

Calculating Method refers to the method a router should use to calculate the exact location of one of its neighboring routers. Due to the Space Predictability Model described earlier, we know that it is possible to determine an object's location at a given point in the future using its six orbital parameters and a specified calculating method. Examples of calculating method would be a formula to calculate the position in orbit of a satellite at time *t* using the formula and the satellites orbital parameters. This would have to assume that the satellite's velocity and path do not change between the time the calculating method and parameters are received and the time represented by *t*. If the velocity or path of the satellite *did* change, it would be its responsibility to notify its neighboring routers of this change in trajectory. Aside from relatively static orbital formulas for satellite-bound routers, another type of calculating method might be a 3D straight-line distance formula with the starting location and the distance traveled (at given velocity *v*). This would be applicable for routers onboard spacecraft. The orbital parameters for routers in this scenario would be much more volatile than their satellite-bound counterparts as the speed and direction of a spacecraft are more likely to change; in fact, a highly maneuverable spacecraft might be changing speed and direction very quickly which might cause problems if there was no throttle put into place on how often the onboard router should update its neighbors. This throttle however would probably cause a spacecraft to become unreachable during periods of frequent maneuvers. This behavior may or may not be acceptable, but I would assume that it should be able to remain in the network even during these periods. It would be interesting to investigate how this sort of functionality might be achieved.

2. SDIP Routing Algorithm

The initial SOSPF research mentions the SDIP algorithm and how SOSPF would be using it instead of the standard Djikstra path-finding algorithm; however, it does not going into much detail as far as how to implement it or integrate it with the existing OSPF code. The implementation, integration and testing of SDIP alone will probably be a significant research topic but is necessary for the eventual implementation of SOSPF (especially if the algorithm has never been implemented before). This portion of the SOSPF project will probably also involve developing a sort of simulation program for SOSPF to test the behavior and performance of SDIP when intermittent links exist in the network. An SOSPF simulator would probably be a worthwhile topic in and of itself as it would be worthwhile to test it that way before spending millions of dollars deploying routers with the software on it into outer space.

6. References:

[1] BANTAN, N., A Routing Protocol and Routing Algorithm for Space Communication, PhD Dissertation, Department of Computer Science, Kent State University, Advisor Javed I. Khan, May 2007.
[2] OSPFD Routing Software Resources page (http://www.ospf.org)