NMSU-ECE-05-002

Annual Report: Research Supporting Satellite Communications Technology

Stephen Horan and Raphael Lyman
Manuel Lujan Space Tele-Engineering Program
New Mexico State University
Las Cruces, NM

Prepared for

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, MD

under Grant NAG5-13189

1 March 2005

Klipsch School of Electrical and Computer Engineering
New Mexico State University
Box 30001, MSC 3-O
Las Cruces, NM 88003-8001

CONTENTS

List of Tables

List of Figures

SUMMARY

This report describes the second year of research effort under the grant "Research Supporting Satellite Communications Technology," NAG5-13189. The research program consists of two major projects: Fault Tolerant Link Establishment and the design of an Auto-Configurable Receiver that are being conducted by faculty and students at New Mexico State University (NMSU).

The Fault Tolerant Link Establishment protocol is being developed to assist the designers of satellite clusters to manage the inter-satellite communications. The protocol design is based on token passing to establish channel access permissions and periodic heartbeat messages to probe for link failures between nodes. The protocol management permits the overall cluster of satellites to be partitioned into subnetworks to maintain connectivity between subsets of nodes based on mutual connectivity. Within each subnet, there is a cluster head to manage the token passing. Subnetworks can also merge to form larger subnets. Inherent within the design is the recognition that these inter-satellite links will occasionally under go corruptions that may make a link unreliable. The protocol is being designed so that momentary corruptions of message traffic will not cause link failures. During this second year, the basic protocol design was validated with an extensive testing program to verify that the protocol states operated correctly. After this testing was completed, a channel error model was added to the protocol to permit the effects of channel errors to be measured. This error generation was used to test the effects of channel errors on Heartbeat and Token message passing. The C-language source code for the protocol modules was delivered to Goddard Space Flight Center for integration with the GSFC testbed.

The need for a receiver autoconfiguration capability arises when a satellite-to-ground transmission is interrupted due to an unexpected event, the satellite transponder may reset to an unknown state and begin transmitting in a new mode. Data will be lost while the ground-station receiver determines the new mode and makes adjustments. To speed the reconfiguration of the receiver, we are developing algorithms that allow the new transmission parameters to be determined automatically by examining the received signal itself. The parameters of interest for the TDRSS Multiple Access Return Service are data rate, data format, and details of the convolutional encoding. We have found that some of these parameters can be determined reliably based on the statistics of the signal alone, while others require the assumption that the data sequence is organized according to some known data-link protocol. During Year 2, the focus of this report, we completed testing of these algorithms when noise-induced bit errors were introduced. We also developed and tested an algorithm for estimating the data rate, assuming an NRZ-formatted signal corrupted with additive white Gaussian noise, and we took initial steps in integrating both algorithms into the SDR test bed at GSFC.

1 Faculty and Students Supported

The research program consists of two major projects: Fault Tolerant Link Establishment and the design of an Auto-Configurable Receiver. The following faculty and staff contributed to the program for the first year:

- Dr. Stephen Horan, Professor of Electrical and Computer Engineering
- Dr. Raphael Lyman, Assistant Professor of Electrical and Computer Engineering
- Mr. Giriprassad Deivasigamani
- Mr. Rahul Vanam
- Mr. Praveen Gopinath Thonour
- Mr. James Rodgers

Mr. Deivasigamani has used this work as the basis for his MSEE thesis project. Mr. Thonour is building upon Mr. Deivasigamani's thesis and will be using this project for his MSEE thesis later in 2005.

2 Fault-Tolerant Link Establishment

2.1 Introduction

This report describes the work performed under the second year of the algorithm development for the cluster-networking algorithm that considers channel errors. The primary emphasis of this year's work was on testing and validating the algorithm developed during Year 1. The full details of the testing are found in [7]. In this report, we will summarize the work performed to develop a link establishment protocol for a network of satellites forming a cluster. This link establishment algorithm will expect that

the inter-satellite channel will be unreliable and will, therefore, need to consider channel errors in the decision making process. The problem of managing satellite-cluster communications links similar to that studied in terrestrial mobile ad hoc networks (MANETS). In examining approaches to this problem, we decided to attempt a realization of an algorithm for routing proposed by Chiang et al. [1]. This particular algorithm was designed for use in fading channels and allows the network of nodes to self-organize into smaller sub-networks. This algorithm was designed for use by 100's of nodes in the network, have a good degree of stability in assigning the roles of cluster head and cluster slave, and allow nodes to move between sub-networks. These are all characteristics of the desired satellite cluster protocol. The Cluster Head election and cluster member-partitioning algorithm is based on a Least Cluster Change (LCC) method to decide to which sub-network a node belongs. The Cluster Head controls the transmission of traffic by use of a token to grant permission to each node for channel access. Sequence numbers are used in the cluster management traffic to eliminate stale information and help nodes synchronize. In the development of the protocol, we use this basic philosophy and augment it with persistence metrics to ensure that simple channel errors do not cause links to be marked as broken or nodes unreachable.

There are competing methods for the routing in ad hoc networks that are being considered by other research groups. The two that are similar, in some respects, to the one considered here are Ad hoc On-Demand Distance Vector (AODV) routing and Dynamic Source Routing (DSR). The AODV protocol is described in [2] and [3] while DSR is described in [3] and [4]. Some of the main differences are between AODV, DSR, and the approach chosen in this study are:

1. AODV and DSR assume no a priori node information while our approach assumes that the network may be pre-seeded with participating nodes,
2. AODV and DSR assume that the networks will be open while our approach assumes that network access will be limited to "trusted" nodes,
3. AODV and DSR use routing caches for routing information while our approach uses a routing table to hold the routing information,
4. AODV and DSR obtain routing information in an on-demand manner while our approach keeps the routing information in a Routing Table which is updated on a periodic basis and the routes are available before the data needs to be sent,
5. AODV and DSR can drop a link due to a single link error while our approach will not declare a link to be down until after several messages have failed,
6. DSR send all of the required routing information in the data package header, while AODV and our approach requires that all intermediate nodes have sufficient path information locally,
7. AODV and DSR assume that the inter-node link range is < 500 m (802.11-type link) while the satellite cluster algorithm needs to cover in excess of 1000 km.

The timing for message passing is illustrated in Figure 2-1**Error! Reference source not found.**. Here, we see the protocols first send a routing request message (RREQ) from the source (SRC) through intermediate nodes (INT) to the destination (DEST). This route request message receives the route reply via a routing reply message (RREP). Finally, the data is sent in the data message. This process is used for the first data transmission or whenever the link has been lost.

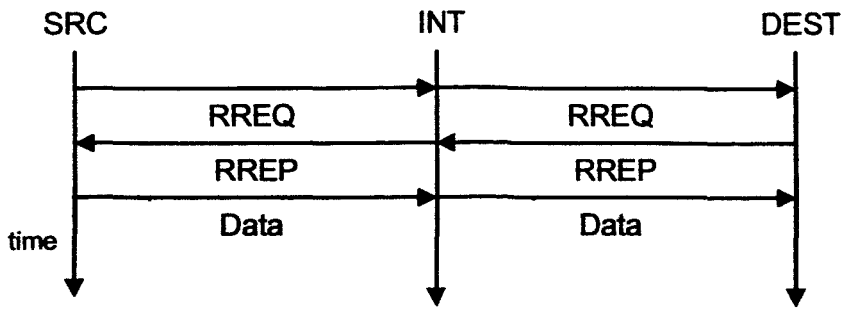


Figure 2-1 – AODV and DSR timing for sending a message.

For nodes in a satellite cluster, several of the characteristics of AODV and DSR would seem to pose problems that a different approach could help remove. In particular, the cluster protocol should

1. Acknowledge that the satellite cluster network will have many, if not all, of its participating nodes known before launch so they can be seeded and not need to be discovered,
2. Do not remove routing information due to channel errors unless they pass a threshold,
3. Do not send whole route lists with every packet but only send routing information updates when the link state changes to keep the routing update bandwidth as small as possible,
4. Use low-bandwidth Heartbeat messages to probe for link failures but send them less frequently than is done on mobile networks where they may be sent approximately every second.

For ease of coding, the initial development of the protocol will not be directly executed in a high-level language such as C. Rather, we will take advantage one of two widely used environments that can be used for the protocol development: Matlab and LabVIEW. The Matlab environment is successfully used in many analysis environments for communications, signal processing, and controls. One addition to the Matlab environment is the *Stateflow* toolkit. *Stateflow* is designed for tasks such as protocol development that can be expressed in terms of states with well-defined transitions. During the fall 2003 semester, the effort was directed towards developing the protocol state diagram in *Stateflow*. While this product does have a large learning curve associated with it, the main deficiency found with *Stateflow* is that it does not directly support networking protocols such as Transmission Control Protocol (TCP) and Unconnected Datagram Protocol (UDP). These need to be developed in other Matlab environments and then run with the *Stateflow* modules. After a number of unsuccessful experiments with *Stateflow* and Matlab, we were not able to devise a successful configuration to make the protocol work with the required networking applications. Therefore, an alternative approach was sought.

In later 2003, the National Instruments released a State Diagram toolkit for use with the LabVIEW programming environment. This toolkit is very similar to the Simulink *Stateflow* toolkit. However, it has one major advantage: the LabVIEW environment fully supports TCP and UDP communications modes without special modification or non-standard modules. Therefore, the language choice for the initial software development was the National Instruments LabVIEW. One major disadvantage of the State Diagram toolkit is that the modules cannot be directly translated by the toolkit into a C-type of

code representation. This limits the portability to those hosts running the LabVIEW environment.

2.2 *Simulation Laboratory Environment*

The test facility at the New Mexico State University is composed of a cluster of seven computers arranged on a common Local Area Network (LAN) segment. The arrangement of the computer cluster is illustrated in **Error! Reference source not found.**. The computers in the cluster are connected to each other over the router segment in the Goddard Annex building where they are housed. The router then provides access to the wider Internet via the NMSU campus backbone. The computers are equipped with a C compiler and the LabVIEW package for protocol development work. The computers use the Windows XP operating system.

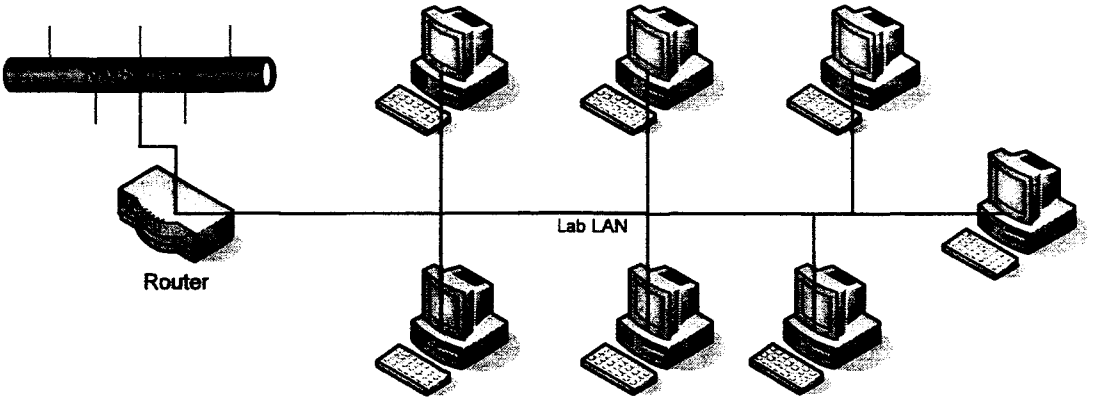


Figure 2-2 – NMSU simulation laboratory configuration.

2.3 *Protocol Specification*

Before generation of the state diagram for the modules was attempted, a detailed protocol specification was codified. The protocol specification details can be found in [5] and they can be summarized as follows:

1. Use a Routing Table pre-seeded with the expected cluster nodes and allow new, trusted nodes to be added later;
2. Use periodic Heartbeat messages to probe the channel for broken links and let the message interval be user-defined and only send the messages to neighbors within one hop;
3. Use a periodic Token passing mechanism to control access within a subnet under the assumption that the overall cluster may need to be partitioned because not every node may be visible to every other node;
4. Send Routing Table updates from a given node to its one-hop neighbors only when that node detects link connectivity changes or it receives better link information from one of its neighbors;
5. Use Cluster Heads to control Token passing within each sub-net where the Cluster Head is defined as that node in the sub-net with the lowest IP address that is one hop away from all members of the sub-net;
6. When a Cluster Head fails or moves away from a sub-net, the survivors determine the next Cluster Head by the one with the lowest IP address.

The message passing protocol developed here is illustrated in Figure 2-3. Here, the heartbeat (HB) messages are used to establish the link connectivity and routing.

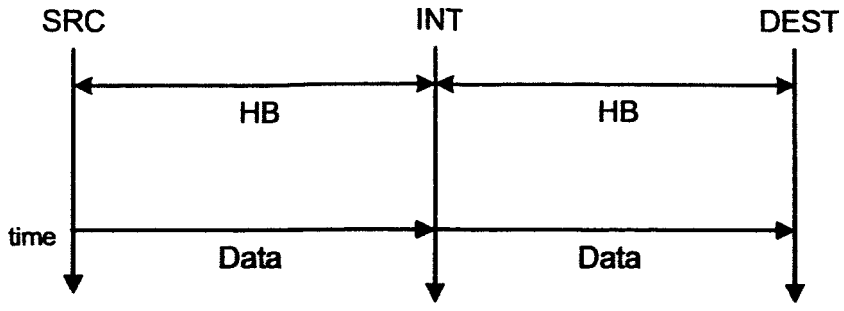


Figure 2-3 – Message passing timing for the protocol developed here.

When data is ready to be sent, the routing table is used and the data is sent directly from the source (SRC) to the destination (DEST) using any intermediate nodes (INT) as necessary.

The detailed specifications were designed as a state machine for realizing the protocol. The top-level states are illustrated in Figure 2-4. In LabVIEW notation, this is the initial Virtual Instrument (VI) defining the protocol. In the INIT state, the user-defined parameters and initial Routing Table and State Table are built to define the protocol variables. Then each node determines if it is a Cluster Head or Cluster Slave based upon the node's IP address and location in the Routing Table. Each node then enters either the HEAD or SLAVE state and executes appropriate processing there. These states may be exited if a state change is detected, for example detecting the failure of the existing Cluster Head, or if the protocol received a management message to stop the protocol.

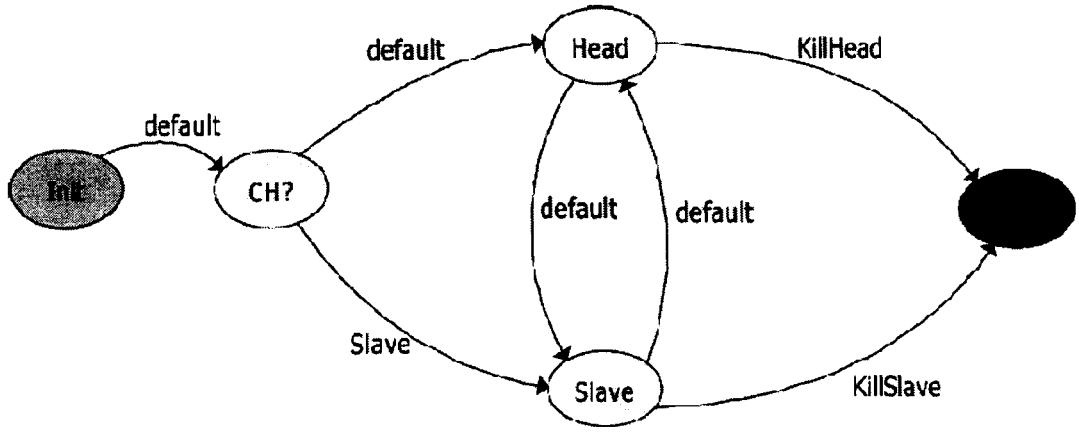


Figure 2-4 -- Top-level states in the cluster link establishment protocol.

The Cluster Head and Cluster Slave have similar state diagrams that are realized as state machines as well and are called as sub-VIs from the main VI. The state diagram for the Cluster Head is given in Figure 2-5. After initialization, the Head and Slave enter a continuous loop. The basic structure is to

1. check for the presence of Heartbeat (HB) messages on the input port and process them if available;
2. check for the presence of Handshake (HS) messages, e.g. a Routing Table update and process them if available;
3. check for the presence of a Token message and process it if available;
4. check for time to issue a Heartbeat message and do so to the one-hop neighbors if it is time;
5. check for the time to issue a Token message and do so to the next entry in the Routing Table for the sub-net.

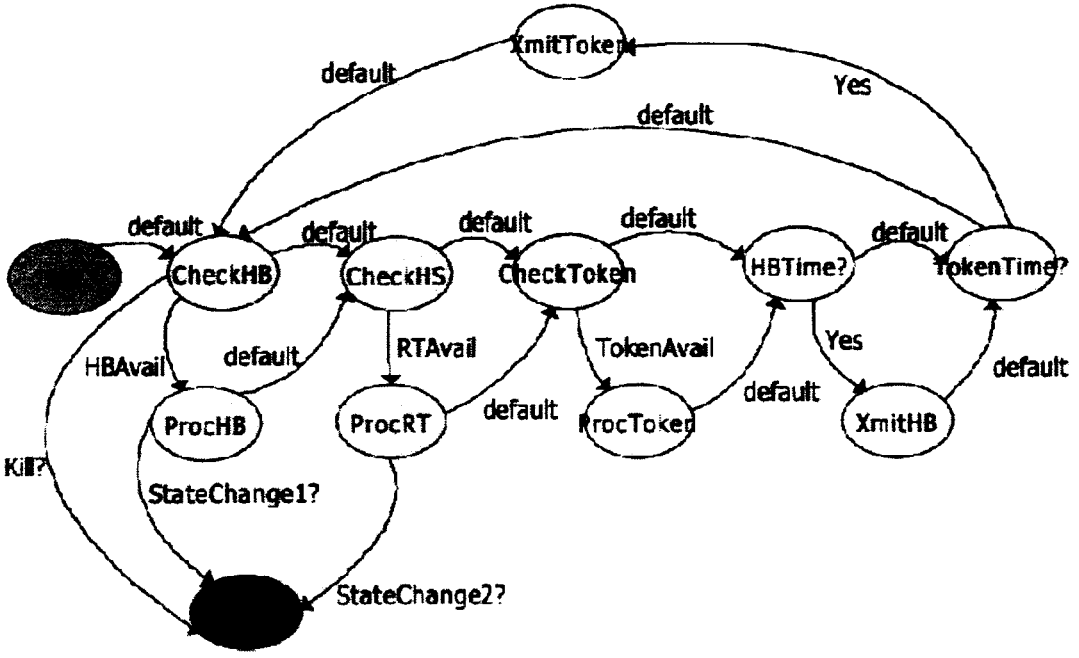


Figure 2-5 – State diagram for the Cluster Head state. The Cluster Slave is identical except for the Token transmission timing.

A Cluster Slave does not issue Token messages so the Token timing check is not part of the Slave VI states.

The individual states in the Cluster Head and Cluster Slave VIs can be made into VIs of their own with a finite number of states. This is illustrated in Figure 2-6 for the Process Heartbeat message state. In this VI, the Heartbeat message of processed and the Routing Table is updated. The Routing Table may also be transmitted to the one-hop away nodes if significant changes are detected as part of the Heartbeat message processing.

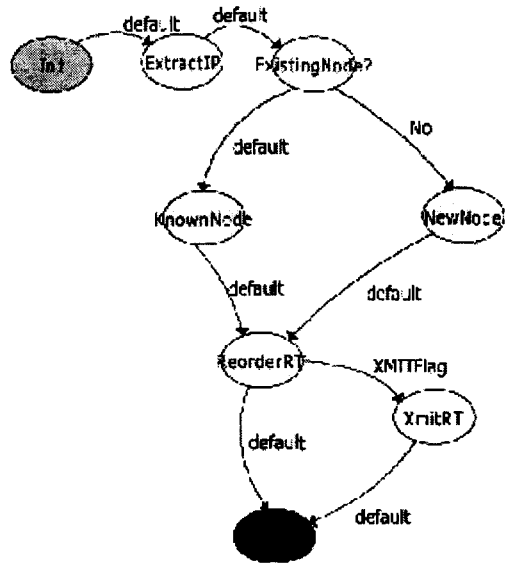


Figure 2-6 -- The process Heartbeat states within the Cluster Head or Cluster Slave VIs.

From this point on, it is frequently possible to encode all of the state processing within a single-state sub-VI rather than making further refinements to the state machine. This is a design decision for the protocol designer. The advantage the State Diagram toolkit brings is that the state diagrams can be developed quickly and then more time can be spent on the detailed processing modules. In the protocol software development, 26 modules were developed to program the protocol. Some of these modules perform the flow control between states within the VIs while others perform actual computations or state variable manipulations.

2.4 Channel Error Generation

To test the effects of channel errors on Heartbeat and Token message passing, a channel error generator was developed. Instead of a bit-wise error generation as had

been used in [8]. With networking packets, one or more errors at any location within the packet will cause the packet to be rejected. For the channel errors here, we will then reject a message that has any form or error. To set the threshold for rejection, we note that the probability of a packet of length N bits being received correctly, P_0, when the channel Bit Error Rate (BER) is p, is given by

$$P_0 = (1-p)^N \tag{2.1}$$

The probability of rejecting a packet, P_R, for any number of errors is then given by

$$P_R = 1 - P_0 \tag{2.2}$$

The algorithm for computing if a packet should be rejected upon reception is based on the following three steps:

- Compute P_0 for a packet based upon its length and the user-specified BER
- Use the system random number generator, which is uniform on [0,1], to generate a random number.
- If the generated random number is greater than P_0, then reject the packet; otherwise accept the packet.

This was validated against a bit-wise procedure for rejecting the packets and found to give the same results to within statistical fluctuations.

2.5 Protocol Testing

The software developed for the link establishment protocol was tested and the full description is given in [6]. The testing philosophy was to build the basic state variable structure, verify that it could be checkpointed to a disk file and recovered, and then add

well-defined modules that built incrementally upon the successful development and testing of previous modules. This is where process reverses flow from the design stage. During the design, we tried to let the protocol logic dictate the state flow within the VIs and defer the detailed processing until as late as possible. Once the detailed processing modules are completed, they are tested at the unit level to ensure proper functionality and then integrated with other modules. Eleven test sequences were run to verify that the modules and VI control logic functioned properly. The full details of the testing are given in [7].

2.6 Test Results

Once basic functionality was established, the protocol was stressed by adding channel errors on the links. We assumed that a radio environment with a synchronous Phase Shift Keying demodulator would be used. In this case, the energy per bit to noise spectral ration, Eb/No, would be used to characterize the link. The user interface was modified to permit this parameter to be selected. Test runs with a channel bit error rate of 0.001 showed that the number of token messages lost due to channel errors matched the expected value based on an assumed white noise error distribution. We also observed that the total probability of token loss scaled with the number of nodes in the cluster.

The test program described in [6] was intended to validate the initial phase of the satellite cluster link-establishment protocol development. This initial development is intended to provide a basic functionality that can be further tested and refined. The capabilities demonstrated in this testing included:

1. The ability to use IP addresses as the means to control node designations as either a Cluster Head or Cluster Slave and have these designations based upon the current contents of the Routing Table. IPv4 is used now but this can be extended to IPv6 in the future.
2. The ability to use the contents of the Routing Table to
    a. determine the path of the Token through the Cluster members,
    b. determine which cluster members are to receive a Heartbeat message from each node,
    c. determine which cluster members have become inactive or unreachable.
3. The ability to transmit Heartbeat messages with a predetermined re-issue period to probe the cluster for unreachable members.
4. The ability to transmit Token messages with a predetermined re-issue period to allow controlled access to the channel.
5. The ability to exchange Routing Table messages between the cluster members and update this Table based upon changing conditions.
6. Persistence in the transmission of Heartbeat messages until the time-out period is exceeded.
7. Persistence in issuing Token messages and nodes are not removed from the Token path until the time-out period has expired.
8. The ability of the Cluster Head to control the issuing of Token messages.
9. The ability of the cluster nodes to select a new Cluster Head if the original Cluster Head fails or becomes unreachable.

After the protocol validation testing was completed, we began the initial set of protocol tests with channel errors. In these tests, the cluster was configured with seven active nodes and a single Cluster Head. The tests were run with a channel bit error rates of 0.001 and 0.000001. The first set of tests was to investigate the perceived cluster partition size from the point of view of the Cluster Head when the Heartbeat messages were subject to link errors. If no Heartbeat messages are received by node j from node k during a specified persistence interval, then node j declares the link from node j to node k to be down. The timing for this event is illustrated in Figure 2-7. In this figure, the initial heartbeat (HB) messages are exchanged successfully. Then, at a later time, three successive message corruptions cause the HB messages from Node k to Node j to be rejected by Node j. Node j then sets the link state to Node k as "unreachable." If any one of the three HB messages had succeeded, Node j would still consider Node k to be reachable. When Node j receives a HB from Node k, the state will return to reachable. In this test, the persistence interval was varied from 3 to 5 times the Heartbeat re-transmission interval to see how frequently the Cluster Head declared nodes to be unreachable due to channel errors. The results are illustrated in Figure 2-8 for the two BER levels. When the channel BER was 10^{-6}, the original persistence interval of three times the Heartbeat re-transmission interval was used. When the BER was 0.001, three different persistence intervals were investigated. In this figure, we see that with a BER of 10^{-6}, the Cluster Head case will occasionally declare a node to be unreachable and the partition size decreases by one. However, the cluster quickly recovers, the node is reintegrated, and the cluster size returns to its normal value of seven members. When the channel BER is 0.001, if we increase the persistence

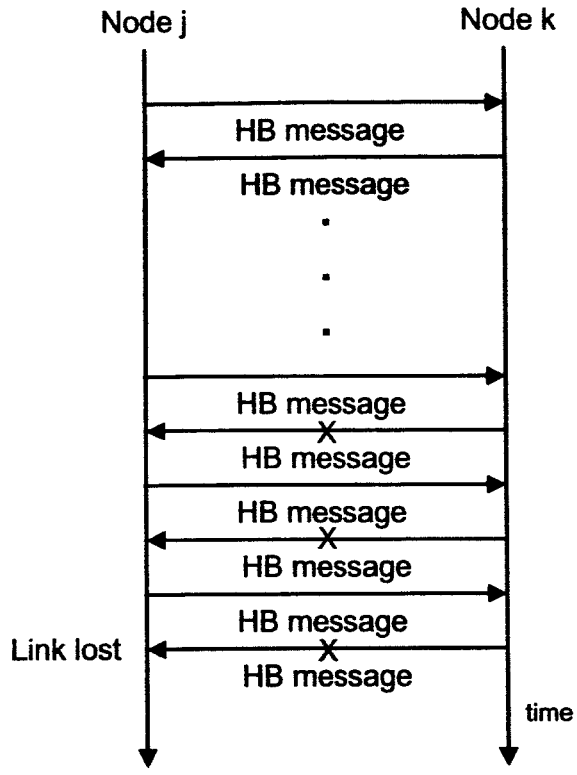


Figure 2-7 – Loss of a link due to corruption of the heartbeat messages from channel errors.

interval to 4 or 5 times the Heartbeat transmission interval then we can achieve results similar to the BER = 10^{-6} results. The penalty for this lack of dropping links is the increased time required for all nodes to know of the new cluster configuration by having all of the nodes receive the updated routing table. When the persistence interval is reduced to three times the Heartbeat interval, then the cluster behaves differently. In this case, the number of nodes that are reachable in one hop decreases throughout the simulation. The nodes are still reachable from the Cluster Head. However, they are forming smaller partitions with multiple hops between the nodes and the original Cluster

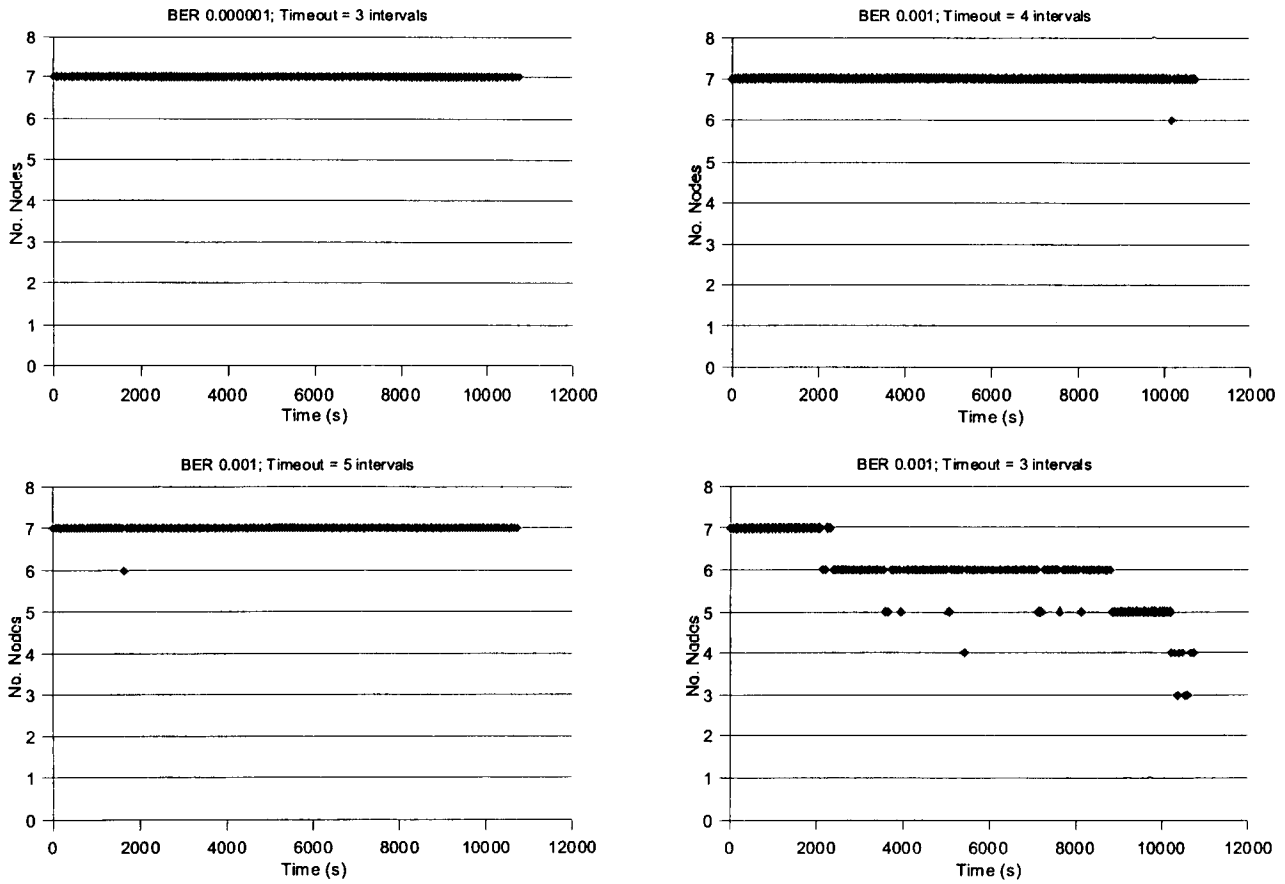


Figure 2-8 -- Number of nodes in the Cluster Head's partition as a function of channel BER and length of persistence interval relative to the Heartbeat re-transmission interval.

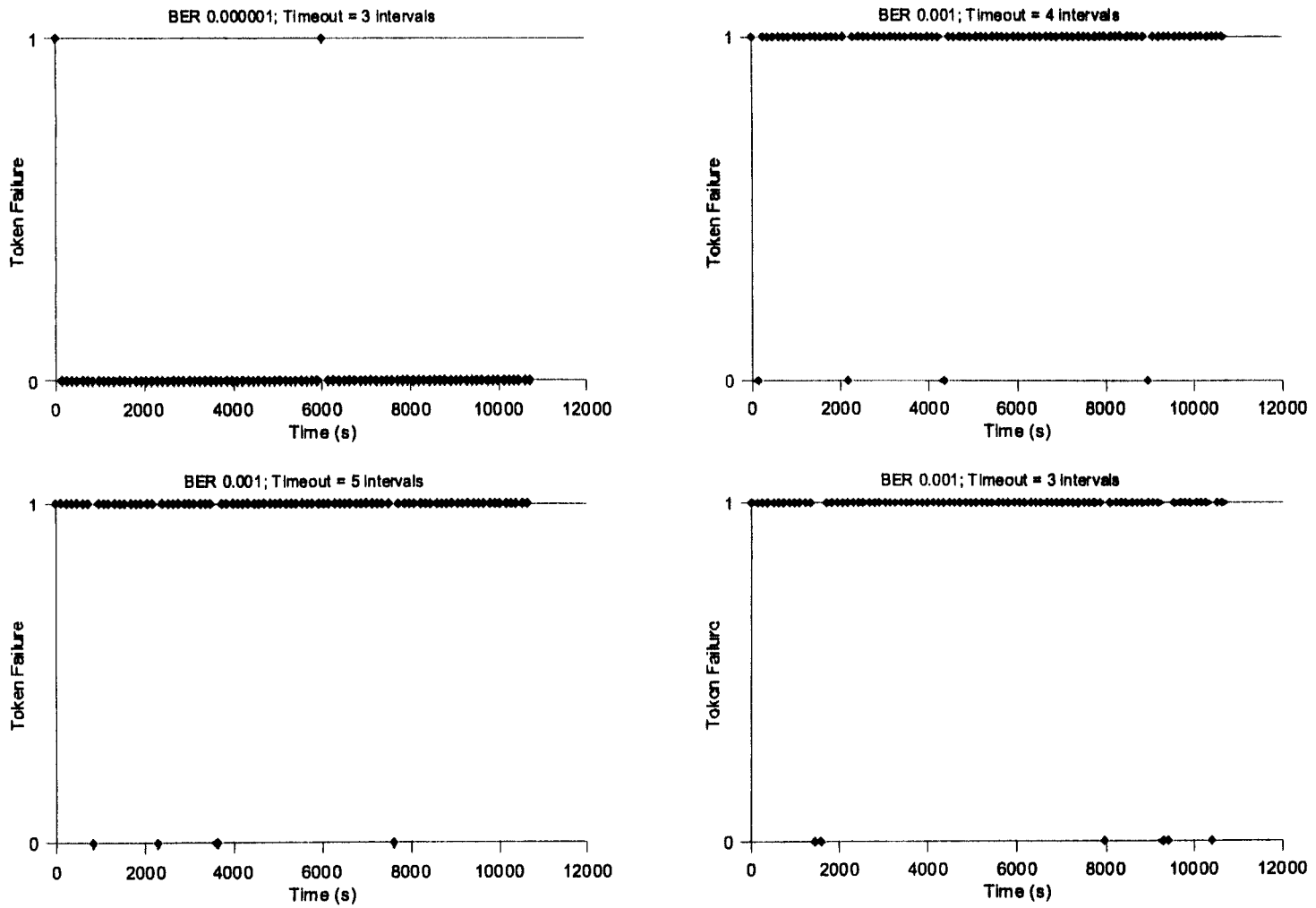


Figure 2-9 --- Token message failures as a function of channel BER and length of persistence interval relative to the Heartbeat re-transmission interval.

Head because they believe that they have lost direct contact with the Cluster Head. While the cluster starts with the original number of nodes, the use of a token to control channel access is seen to have problems if the token message does not have some form of intrinsic error correction added. This behavior is shown in Figure 2-9 for the same channel error and Heartbeat interval cases as investigated above. Here, a channel BER of 10^{-6} causes a minimal number of token message losses over the simulation interval. The initial token loss at the start of the simulation is merely the result of not having the cluster fully integrated at that time. As can be seen for the cases with a BER of 0.001, the change in the persistence interval does not substantially affect the loss of the token messages. In these cases, the routing tables have paths to all nodes. However, the channel errors prevent the token from traversing the entire cluster. Table 2-1 gives the probability of token message loss for each of these cases.

Table 2-1 -- Probability of Token Message Loss as a Function of Channel BER

BER	Intervals	P(loss)
0.000001	3	2.2%
0.001	5	95.5%
0.001	4	95.5%
0.001	3	93.3%

2.7 Conclusions from the Testing

The LabVIEW State Diagram toolkit can be used to generate VIs that are embodiments of state diagrams as well as having VIs that perform more traditional computational tasks. The largest advantage seen in this process was the ability of these types of state variable toolkits to be good vehicles for organizing the logic flow between states in the protocol and their abilities to be easily edited to modify the logic if flaws are found or if

different approaches are desired. While this type of development could be performed in a high-level programming language such as C, the use of a graphical toolkit made the development process much easier. Because the toolkits also perform syntax checking as the code is developed, they are expected to have a quicker development cycle by eliminating those types of errors.

The first set of results show that there is a definite impact of channel errors on the protocol's performance. While the cluster members can still communicate with each other, the channel errors may fool some nodes into thinking that other nodes are multiple hops away and not directly connected when the channel BER is high. It is expected that this can be mitigated by applying a error correcting code or adding some form of explicit acknowledgment to the token and heartbeat messages. There is a trade-off here between processing overhead and message passing delay. Further experiments need to be conducted to investigate these issues.

2.8 Research Questions

In developing and testing the link establishment algorithm we have uncovered several research issues that need further investigation as the protocol is brought to maturity. These questions include

- How should gateways be configured to expedite data transmission among sub-cluster members
- What is the "best" method for Token Message passing, especially considering channel errors may corrupt the Token Message

- Is there a reasonable way to allocate priority to the data traffic to allow proper control of the shared bandwidth
- What is the appropriate data model for the traffic
- How can multicast addressing be effectively used.

These issues are discussed further in the subsections below.

2.8.1 Gateways

The original Least Cluster Change algorithm in [1] offered three mechanisms for addressing the routing between sub nodes in the overall cluster. The original LCC algorithm defined subnets based upon that set of nodes within a single hop of the cluster head. A large collection of nodes would then self-partition into sub-networks where each node would be identified as being identified with at least one cluster head with a single-hop link to the head. Naturally, there is the possibility that a given node could be within one hop of multiple cluster heads. In this case, these nodes would become getways between sub-nets. The ways of routing a message through subnets that were proposed by the LCC developers are illustrated in Figure 2-10. In all three cases, a message needs to be routed from Node 1 to Node 11. Nodes 1, 6 and 10 are cluster heads for their respective sub-nets. Nodes 5, 7, and 11 are gateway nodes since they are common to multiple sub-nets (the other sub-net containing Node 11 is not shown for illustration clarity). The proposed routing paths are as follows:

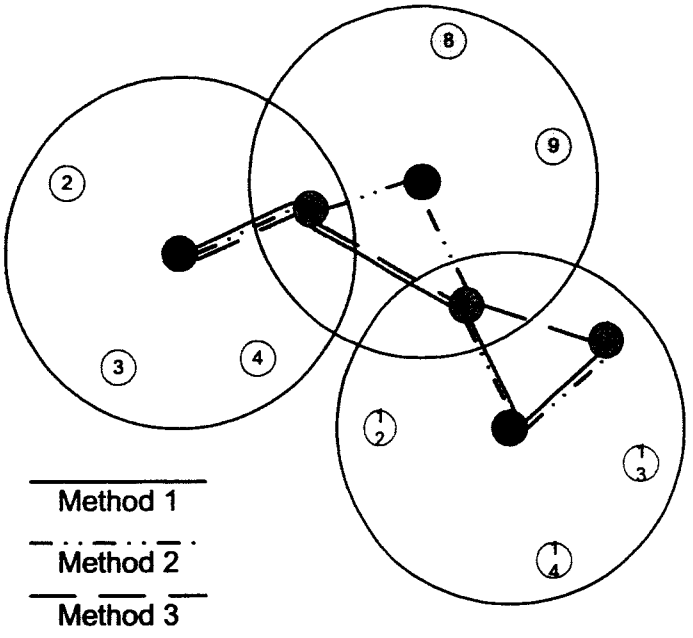
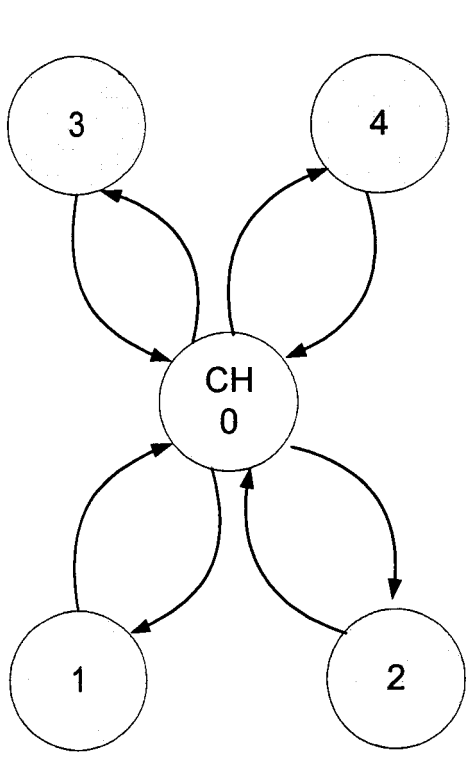


Figure 2-10 – Three proposed paths for message routing through gateways.

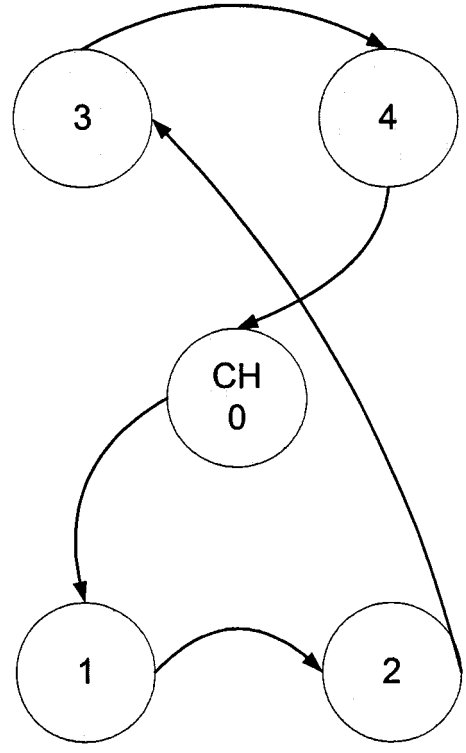
1. Method 1: CH1 to GW5 to GW7 to GW10 to Node 11
2. Method 2: CH1 to GW5 to CH 6 to GW7 to CH10 to Node 11
3. Method 3: CH1 to GW5 to GW7 to Node 11

The question to be resolved with this is which method is most efficient (least end-to-end latency)? The reason why the answer is not obvious is that the message passing speed is a function of the token availability in the sub-nets. In the original LCC method, the cluster heads hold the tokens more often than the nodes in the sub-net hold the token. Therefore, routing through the cluster head may have some speed advantage in those cases. However, we have modified the protocol to pass the token in IP order so the cluster head may not have an intrinsic advantage. The research question here is to

look at the interaction of the message passing method and see what is the most efficient way given the potential token passing methods.



Token Passing in the "Least Change" Algorithm



Token Passing in the "NMSU" Algorithm

Figure 2-11 – Token passing methodologies for the LCC algorithm and the NMSU variation to the LCC algorithm.

2.8.2 Token Passing Mechanism

The original development of the LCC algorithm uses a cluster-head controlled token passing mechanism as illustrated in Figure 2-11. Here, the cluster head issues the token to each node specifically. The node holds the token for a specified time and then

returns it to the cluster head. The cluster head then moves to the next node in sequence and issues that node a token. This process is repeated until all nodes have had the token once. In this method, the cluster head holds the token more than any individual node because the token is always passed back to the cluster head. In the NMSU modification to the LCC algorithm the token passing is passed based upon IP order in the cluster. In this method, the cluster head holds the token just as frequently as each individual node. We believe that this is a fairer method if the cluster members are homogeneous satellites. There are three research questions to be answered with the token passing method:

1. If the token message is unprotected by a FEC, is the original LCC method a better way to pass the token so that the probability of losing the token due to channel errors is smaller?
2. What is the lower bound on token rotation time in the cluster?
3. Should the sending of Heartbeat and Routing Table messages be tied to holding the token?

One may suspect that the IP-based token passing might always be quicker than the cluster head controlled method. However, as illustrated in Figure 2-12, we can see that the NMSU method permits nodes within the cluster to act as routing nodes. In this case, there is a delay in the message passing as the routing function happens. As we have seen in the testing so far, token loss is a problem at realistic channel BER's. Since this methodology is supposed to have immunity to channel errors, the token passing method becomes important with channel errors. Since FEC represents a

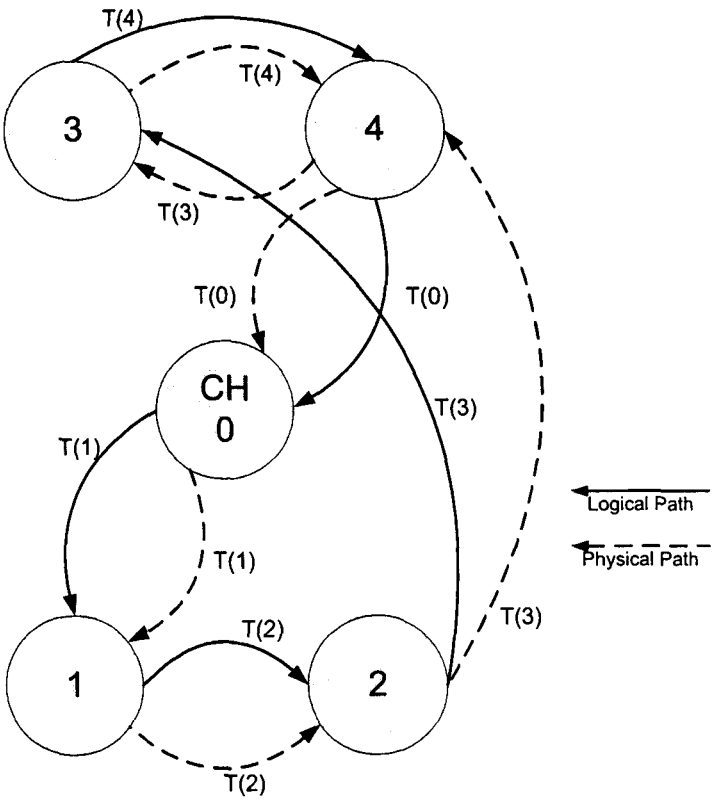


Figure 2-12 – Physical versus logical path routing of a message.

processing overhead, it may be possible to have quicket token passing without the FEC if the cluster head closely manages the token passing. Since lost tokens will affect the ability of the cluster members to efficiently pass data, control of the token and preventing loss is important.

The lower bound on the token rotation time will affect the ability of instituting a priority mechanism and/or computations of token hold time. Both of these issues are design features of terrestrial token LAN architectures and are expected to be useful here as well.

The investigators are looking into the possibility of using the multicast address block to transmit both Heartbeat and Routing Table messages. In the multicast channel, one would need a means to avoid collisions. One method for doing this is to tie the Heartbeat and Routing Table message sending to holding the token. The research question is does tieing the Heartbeat and Routing Table transmission provide an improved performance over not providing that level of control?

2.8.3 Priority Allocation

To provide fair and efficient data passing, the NMSU investigators envision the need for a cluster priority allocation mechanism. This could be used by the cluster head to modify token holding times or provide a way to rapidly pass highly senstivie traffic. An associated issue is to provide the ability to provide multiple tokens on parallel data channels. These channels might be in code space as is used with the GPS system. In this mode, what are the appropriate priority mechanisms, how would the tokens be accocated, and how would the bandwidth be allocated? These are all areas that will interact with the data model and other management issues.

2.8.4 Data Model

To investigate the link establishment protocol in more detail, a standard data model for the cluster members is needed. NMSU requests assistance from NASA in this to ensure that the data model is current with NASA mission concepts. The types of data we expect to see in this model are given in Table 2-2. These represent classes of data

with characteristics. However, NMSU needs design test cases to place numbers on the qualitative metrics.

Table 2-2 -- Parameters for cluster data model.

Mission Requirement	Link Type	Bandwidth	Latency	Priority
Exchange status information (position, pointing, sensor status, etc.)	Multicast	Low	Low	High
Exchange mission data sets (camera output, etc.)	Point-to-point	High	Delay acceptable	Low
Exchange routing information	Multicast	Low; asynchrounous generation	Low	High
Ground contact	Point-to-point	High	Low (scheduled)	High

2.8.5 Multicast Addressing

As has been mentioned above, there is a desire to use multicast techniques on the channels. It is expected that this will assist cluster management especially for cases where a new node enters the cluster and needs to announce its presence to the membership and for transmission of Routing Table messages. The investigators will be looking at efficient means for doing this and testing the concepts in the NMSU laboratory. The method will require a further understanding of how these multicast address blocks can be managed for use in the protocol.

2.9 Year-Three Program

The third year of the program will concentrate on answering as many questions as possible from the list of research questions mentioned above. In time order, we propose the following work plan:

1. Evaluate token passing mechanisms in the present of channel errors to determine which of the methods works most efficiently.
2. Determine the lower bound on token rotation time with the different methods.
3. Evaluate tieing Heartbeat and Routing Table message passing to holding the Token.
4. Investigate the gateway message passing issues to determine if there is a preferred method in this protocol.
5. Design a method for priority allocation and try to quantify the data model so that these issues can be incorporated into the protocol.

We will continue investigating the multicast addressing issues. From our initial investigations, we are still attempting to determine the best way of making this happen on the laboratory LAN with our current routers.

As code enhancements are developed, they will be sent to GSFC for inclusion with the SDR laboratory.

2.10 Dissemination of Results

The following papers were generated to describe this research:

1. S. Horan and G. Deivasigamani, "Design of a Fault-Tolerant Satellite Cluster Link Establishment Protocol," *Proc. IEEE Aerospace Conference*, Big Sky, MT, March 2005
2. S. Horan and G. Deivasigamani,, "Using Labview To Design A Fault-Tolerant Link Establishment Protocol," *Proc. International Telemetering Conference*, San Diego, CA, October 2004.
3. S. Horan and G. Deivasigamani, "Design of a Fault-Tolerant Link Establishment Protocol," *Space Internet Workshop IV*, Goddard Space Flight Center, June 2004.

The Master's Thesis "Design of a Fault Tolerant Link Establishment Protocol for Satellite Clusters," was produced by G. Deivasigamani in November 2004.

2.11 References

[1] C-C Chiang, H-K Wu, W. Liu, M. Gerla, "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel," IEEE Singapore International Conference on Networks, 1997, p. 197 - 211.

[2] C. E. Perkins, E. M. Royer, I. D. Chakeres, "Ad hoc On-Demand Distance Vector (AODV) Routing," Internet Draft draft-perkins-manet-aodvbis-00.txt, October 2003.

[3] C. E. Perkins, E. M. Royer, S. R. Das, and M. K. Marina, "Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Networks," IEEE Personal Communications, Feb. 2001, 16 – 28.

[4] J. Broch, D. B. Johnson, and D. A. Maltz, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks," Internet Draft draft-ietf-manet-dsr-01.txt, December 1998.

[5] S. Horan and G. Deivasigamani, "Link Establishment Algorithm Development – Phase I," NMSU-ECE-04-004, Las Cruces, NM, May 2004.

[6] S. Horan and G. Deivasigamani, "Link Establishment Algorithm Development – Test Report," NMSU-ECE-04-005, Las Cruces, NM, May 2004.I

[7] G. Deivasigamani and S. Horan, "Masters Thesis: Design Of A Fault Tolerant Link Establishment Protocol For Satellite Clusters," NMSU-ECE-04-007, November 2004.

[8] S. Horan and R. Wang, "Design of a Space Channel Simulator Using Virtual Instrumentation Software," *IEEE Trans. Instrument and Measurements*, Vol. 51, No. 5, October 2002, p. 912-916.

3 AUTO-CONFIGURABLE RECEIVER

3.1 Introduction

The goal of the Auto-Configurable Receiver project is to develop signal-processing algorithms that will allow key space-to-ground transmission parameters to be estimated from the received signal itself. Such a capability would be useful, for example, when an unexpected event causes a satellite transponder to reset and begin transmitting in a new mode. It could also be important in managing communication between multiple disparate deep-space assets [1]. As has been previously reported [2], the parameters to be estimated, drawn from the TDRSS Space Network Multiple-Access Service [3], fall into two categories. The first includes parameters that may be estimated from the statistics of the signal itself, such as the data rate and the data format (e.g., NRZ vs. BiΦ). The second category consists of parameters that are more easily estimated if frame-formatting information is used. These parameters include the data-format variant (e.g., NRZ-L vs. NRZ-M), the convolutional coding rate, the bit inversion pattern after convolutional encoding, and the node synchronization (see Lyman et al. [2], for a more complete description of the parameters). During Year 1 of the project, we developed algorithms for estimating parameters in the second category, and we tested under noise-free conditions. During Year 2, the focus of this report, we completed testing of these algorithms when noise-induced bit errors were introduced. We also developed and tested an algorithm for estimating the data rate, assuming an NRZ-formatted signal corrupted with additive white Gaussian noise, and we took initial steps in integrating both algorithms into the SDR test bed at GSFC.

In the following sections, we discuss first the test results for the frame-format based estimation algorithm, then we discuss the data-rate estimation algorithm and its test results. Finally, we outline our plans for the third year of this project.

3.2 Frame-Format Based Estimation

For some of the transmission parameters of interest, our estimation algorithm assumes that the transmitted data has been formatted according to one of a given set of link-layer protocols, such as HDLC [4], or the CCSDS TM protocol [5]. As discussed in the previous section, the parameters that we are currently estimating this way include the data-format variant, the convolutional coding rate, the bit inversion pattern and the node synchronization. The details of this algorithm have been described elsewhere [2]. In summary, the procedure assumes that the received signal has been demodulated. It then decodes the demodulated data sequence using assumed values for the parameters to be estimated. If the resulting decoded sequence is properly formatted according to one of the given protocols, then the assumed parameter values are declared to be correct.

We have developed a Matlab simulation that exercises this algorithm. Under each of the possible frame structures, the simulator generates 1000 random data sequences encoded under each of the possible assumptions about the transmission parameters. Noise is modeled by introducing bit errors, with a given probability, into each sequence, and then the sequence is fed to the parameter-estimation algorithm. The estimates returned by the algorithm are compared with the assumptions used in encoding the

sequence. If one or more parameters were estimated incorrectly, an error is recorded, and the simulator state giving rise to the error is logged.

The random data sequences are generated using simple models of the assumed protocols. When the CCSDS protocols are assumed, for example, the sequences consist of a fixed number of frames, each with 200 bits of random user data prefixed by an appropriate sync word. This is done because proper formatting of data frames under these protocols is determined by checking for the presence of a known sync word. This sync word varies depending upon the rate of turbo coding applied. Our algorithm checks the decoded sequence for the presence of each of the possible sync words, but the turbo encoding itself does not affect the estimation procedure. Thus, when we model a turbo-coded sequence of a given rate in the simulation, we apply the appropriate sync word, but we do not actually encode the random data. The first frame of each sequence is truncated at a random point, so that the estimation algorithm does not know the starting position of the data. The HDLC frames are generated similarly, except that a frame-check sequence is appended to the user data, and then a bit-stuffing operation is applied, as described previously [2], before the flag byte is attached.

The simulations were carried out for the following parameter variations:

1. **Data-link protocols:** HDLC, CCSDS-TM with and without turbo codes. The turbo code rates include 1/2, 1/3, 1/4 and 1/6.
2. **Data-format variant:** -L, -M or -S.
3. **Number of frames:** 5, 6, 7.
4. **Bit error rate:** HDLC: 2×10^{-5} to 6×10^{-4}, CCSDS: 1×10^{-4} to 1.5×10^{-3}.

5. **Convolutional coding rates:** uncoded and rate ½
6. **Inversion pattern:** with and without inversion of every other bit
7. **Node synchronization:** Number of nodes tested equals the inverse of the coding rate.

Figures 3-1 through 3- 6 are plots showing the parameter estimation error rate vs. probability of received symbol error for different data link protocols and numbers of frames, as mentioned above. The error rates correspond to the maximum of the errors encountered among the variations of parameters 2, 5, 6, and 7 above. From these plots, it is observed that error rates decrease as the length of the data increases. In particular, the number of transmitted frames is very important.

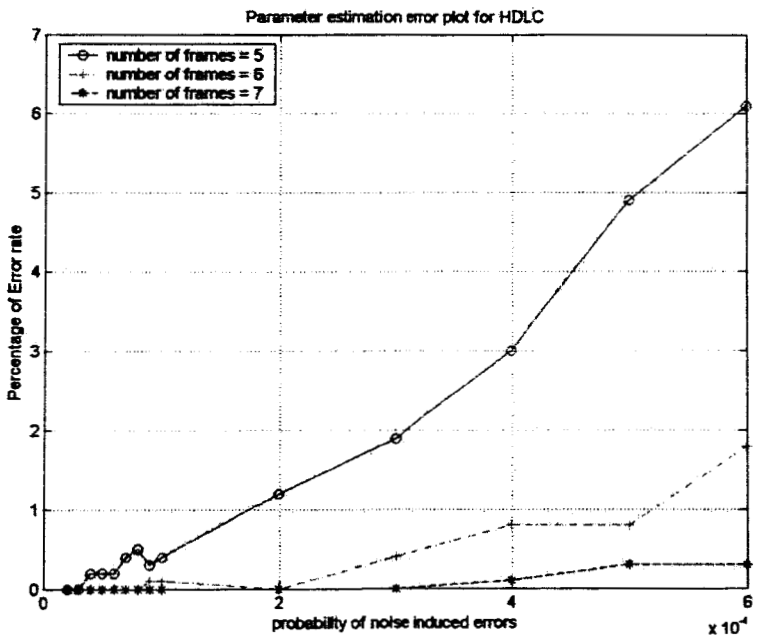


Figure 3-1 – Estimation error rate vs. probability of symbol error for HDLC.

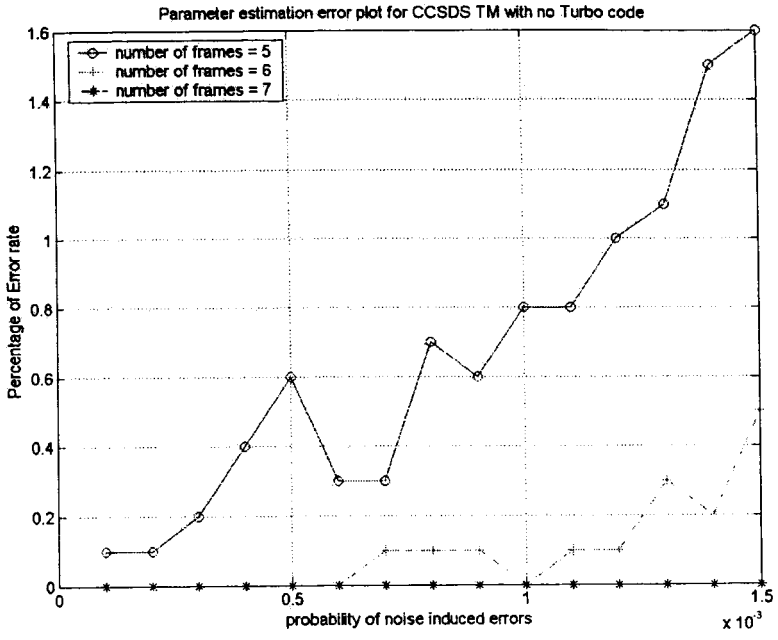


Figure 3-2 -- Estimation error rate vs. probability of symbol error for CCSDS with no Turbo code.

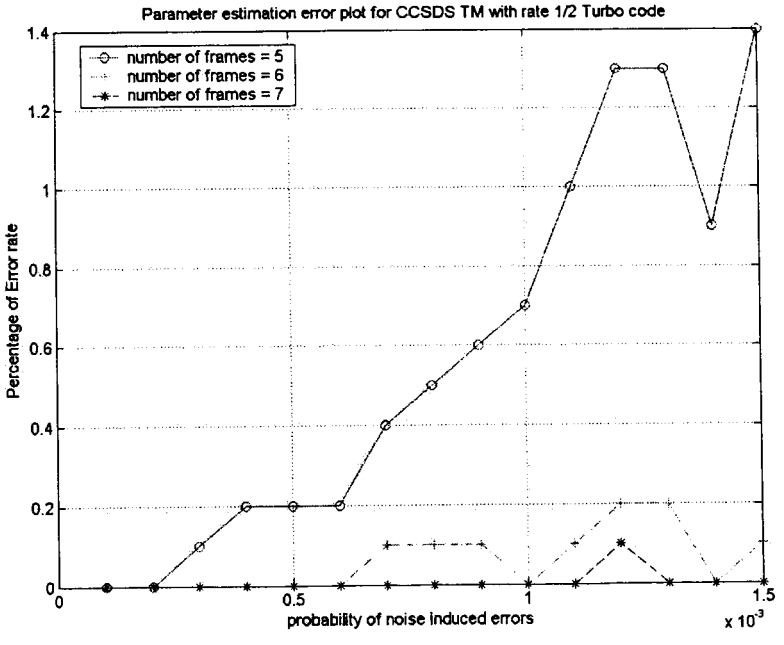


Figure 3-3 -- Estimation error rate vs. probability of symbol error for CCSDS with rate 1/2 Turbo code.

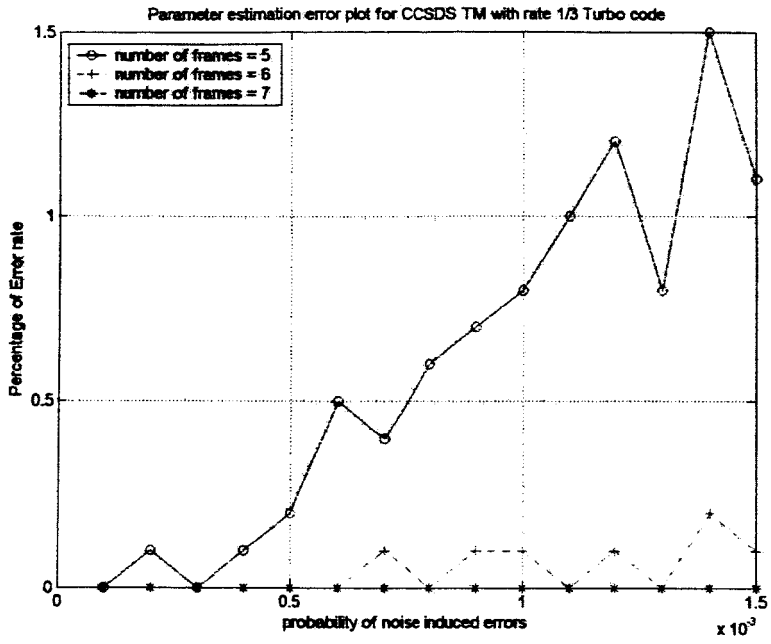


Figure 3-4 – Estimation error rate vs. probability of symbol error for CCSDS with rate 1/3 Turbo code.

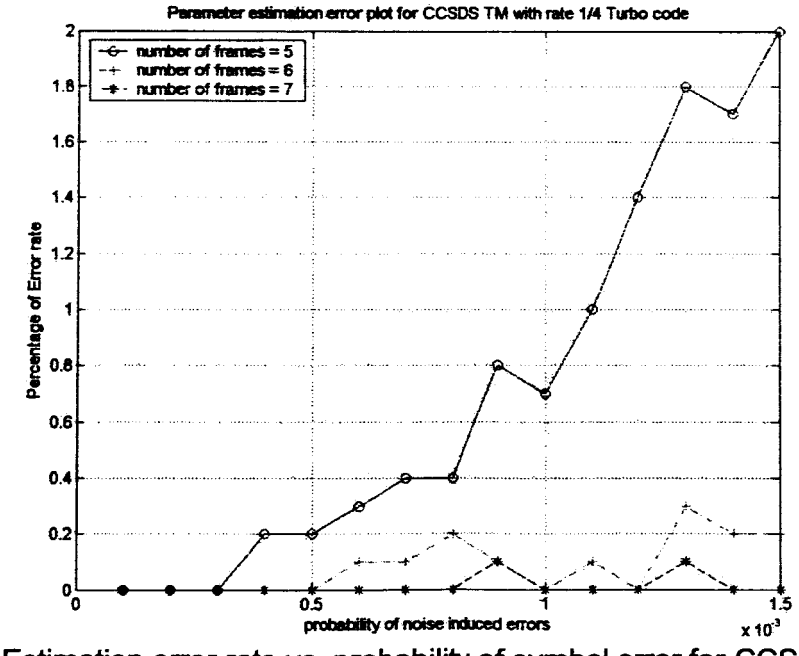


Figure 3-5 – Estimation error rate vs. probability of symbol error for CCSDS with rate 1/4 Turbo code.

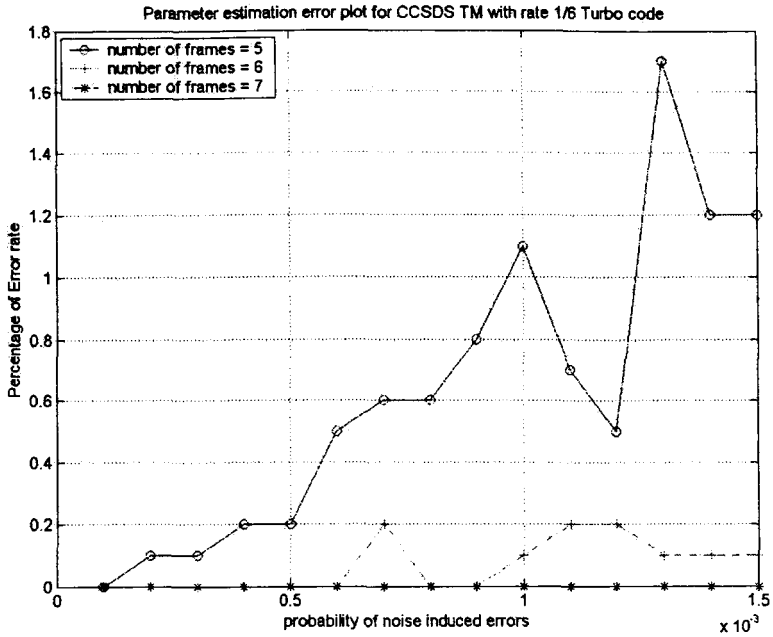


Figure 3-6 -- Estimation error rate vs. probability of symbol error for CCSDS with rate 1/6 Turbo code.

3.3 Data-Rate Estimation

As was mentioned in the introduction of this section, data rate and data format can be estimated without making use of frame-format information. For estimation in a noisy environment, we have chosen to focus first on data-rate estimation of an NRZ-formatted signal. Our intention is to make this the first Auto-Configurable Receiver module to be integrated into the SDR test bed at GSFC.

We assume that the received signal is a sampled NRZ waveform with a square pulse shape. The symbol rate is R_b, and the symbol interval is $T_b = 1/R_b$. If the waveform were modulated with the binary sequence "`...01010101...`", it would consist of a square wave with pulses alternating between –1 and 1, each pulse having a width T_b.

For more complicated sequences, the pulse widths would vary. For example, the waveform representing "010101110101" would include one pulse of width $3T_b$, corresponding to the segment "111" within the sequence. Still, if the binary sequence is completely random, pulses of width T_b would occur more frequently than pulses of width $2T_b$, $3T_b$, etc.

Assuming that the sample rate of the NRZ waveform is much higher than the symbol rate, we can estimate R_b by stepping along the waveform and detecting the times at which signal-level transitions, from –1 to 1 or 1 to –1, occur. The time between successive transitions is the width of one pulse. Thus, we can examine the transition times and determine the width of each pulse that occurs in the analog waveform. From this information we can build up a histogram of these pulse widths. We expect this histogram to have a large peak at T_b, a somewhat smaller peak at $2T_b$, etc. Thus, as an estimate of T_b, we choose the value of the histogram bin with the largest number of hits. To get the data-rate estimate, we take the reciprocal of our estimate of T_b.

This approach works well in a noise-free environment. When Gaussian noise is added, though, the noise spikes will occasionally cause brief spurious zero-crossings in the analog waveform. The procedure described above would interpret the noise spikes as very short pulses. If the noise level is high enough, these short pulses may occur more frequently than pulses of width T_b. Thus, the histogram would have a misleading peak at a small bin value, and the resulting data-rate estimate would be much higher than the true data rate.

This problem could be addressed by filtering the analog waveform to reduce the number of spurious zero-crossings before looking for transitions. One way to do this is to

convolve the waveform with a square pulse, thus averaging out the large noise effects. The longer the width of the convolving pulse, the more noise reduction is achieved, but if the pulse is much longer than the true symbol interval, then the signal itself is averaged out and the signal transitions become more difficult to detect.

The choice of an appropriate convolving pulse width is thus very important. Unfortunately, we have found that no single pulse width works well for all the data rates of interest. For this reason, we convolve the analog waveform with a set of pulses, each of a different length, resulting in a distinct filtered waveform for each convolving pulse. After each convolution, we perform the transition-detection operation on the filtered waveform, and build up a histogram, as described before. We thus end up with one histogram for each convolving pulse width. We may then compare these histograms to determine which of them contains the most valid information about the true symbol interval.

As an example of how this may be done, recall that in the noise-free case the histogram has sharp peaks at T_b, $2T_b$, $3T_b$, etc. Between these peaks are a large number of bins with no hits at all. When the signal is dominated by noise though, the histogram has a high peak at a very small pulse width. The histogram tapers off, apparently exponentially or hyperbolically, for larger pulse widths, but the number of empty bins is usually small. Thus, from the set of histograms built up, we choose the one with the largest number of zero entries and base our estimate on that. This is the approach we use on the current release of the data-rate estimation software.

Figure 3-7 shows typical performance of the algorithm under preliminary testing using Matlab. We see that for E_b/N_0 of 17 dB and greater, the root-mean square estimation error is held below 3%.

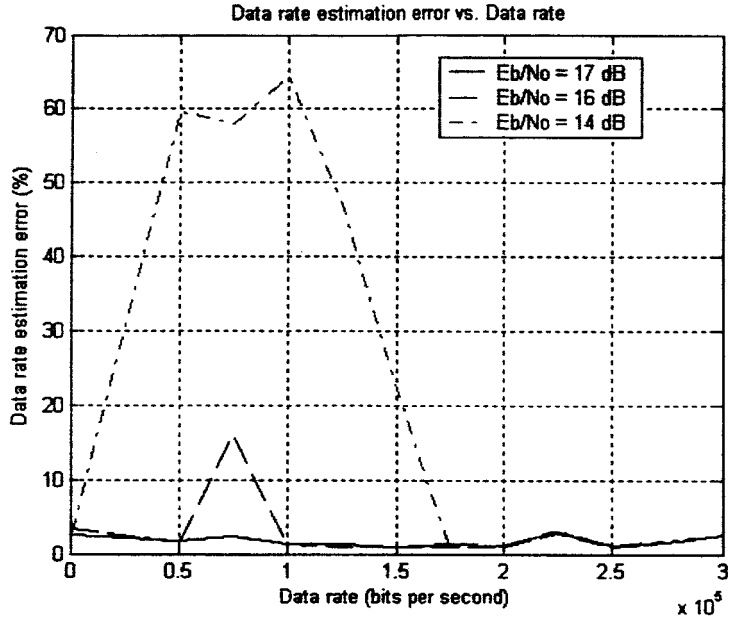


Figure 3-7 – Performance of the data-rate estimation algorithm.

Better performance can be achieved by using a more refined statistical characterization of the histogram. Since a valid histogram has a high, narrow peak centered on T_b, we can measure the standard deviation of the data in the vicinity of the detected peak, and choose the histogram with the smallest such deviation. Preliminary results from Matlab simulations indicate that this approach works well for data with E_b/N_0 of 10 dB and greater. Thus, the prospect for improved performance looks good as we incorporate new approaches into the released software.

3.4 Year-Three Work Plan

Our work plan for the third year of this project includes the following items:

1. Continue to work with the team at Goddard Space Flight Center to integrate the data-rate estimation code module into the SDR test bed (May 31).
2. Use feedback from GSFC to refine the algorithm and user interface (August 31).
3. Improve data-rate estimation performance, holding estimation error below 3% for E_b/N_0 of 8 dB and greater (May 31).
4. Extend the data-rate estimation algorithm to jointly estimate data rate and data format (August 31).
5. Pursue a maximum-likelihood analysis of the joint data-rate/data-format estimation problem to explore opportunities for improving the performance of the current algorithm (December 31).
6. Convert frame-format based estimation algorithms from Matlab to C (August 31).
7. Integrate C-language frame-format algorithms into the SDR test bed (December 31).

3.5 Dissemination of Results

Results of this study will be disseminated by means of conference paper submissions. We expect to be able to submit results from the data-rate estimation work some time during spring 2005.

3.6 References

[1] J. Hamkins, M. Simon, S. Dolinar, D. Divsalar, and H. Shirani-Mehr, "An Overview of the Architecture of an Autonomous Radio," IPN Progress Report 42-159, http://ipnpr.jpl.nasa.gov/progress_report/42-159/title.htm

[2] R. Lyman, Q. Wang, P. De Leon, and S. Horan "Transmission Parameter Estimation for an Autoconfigurable Receiver," *IEEE Aerospace Conference*, March 2004.

[3] *Space Network Users' Guide*, Rev. 8, Mission Services Program Office, NASA Goddard Space Flight Center, Greenbelt, Maryland, 2002.

[4] U. D. Black, *Data Link Protocols*, Englewood Cliffs, NJ: Prentice-Hall, 1993.

[5] *TM Synchronization and Channel Coding*, CCSDS 130.0-R-1, Consultative Committee for Space Data Systems, 2002.

Appendix A. – Link Establishment Protocol C Code Listing

The following listing is the C code version of the link establishment protocol. This code was submitted to GSFC in December 2004.

```
/*=============================================================================
*/
/* Filename          : proto_header.h
*/
/* Definition        : Header file for protocol.c
*/
/*=============================================================================
*/

/* List of Macros */

#define MAX_NODES        10       /* Maximum number of nodes in the network */
#define HSTX_PORT        5000     /* Handshake message Transmit port */
#define HSRX_PORT        5001     /* Handshake message Receive port */
#define HBTX_PORT        5010     /* Heartbeat message Transmit port */
#define HBRX_PORT        5011     /* Heartbeat message Receive port */
#define DATATX_PORT      5020     /* Data Tranmit port */
#define DATARX_PORT      5021     /* Data Receive port */
#define TOKENTX_PORT     5030     /* Token message Transmit port */
#define TOKENRX_PORT     5031     /* Token message Receive port */
#define HBINTERVAL       60       /* Heartbeat message interval in seconds */
#define TKINTERVAL       30       /* Token message interval in seconds */
#define MAXIPLEN         20       /* Maximum length of IP address */
#define HBID             "E"      /* Header(Identifier) for Heartbeat Message
*/
#define RST              "B"      /* Header(Identifier) for RESET Message */
#define LSTGOOD          "C"      /* Header(Identifier) for Last-Good Message
*/
#define RTTBL            "D"      /* Header for Routing Table Message */
#define ACKID            "A"      /* Header for Acknowledgement Message */
#define MAXBUFSIZE       2048     /* Maximum Buffer size */
#define MSGBRK           "Q"      /* Message break character */
#define SERVER_PORTS      3       /* Number of server ports */


/* typedefs */

typedef struct RoutingTable RT;
typedef struct StateTable ST;

/* Structure for Routing Table */

struct RoutingTable
{
```

```
    unsigned long int     ID;        /* IP address */
    int                   no_hops;   /* Number of hops */
    unsigned long int     nextHop;   /* IP address of next hop node */
    unsigned long int     seq_no;    /* Sequence number */
    int                   flag;      /* Flag for sequence number roll-over */
    unsigned long int     timer;     /* Time a message was received */
};

/* Structure for State Table */

struct StateTable
{
    unsigned long int     my_ID;                /* IP address */
    unsigned long int     iptable[MAX_NODES];   /* List of IP addresses */
    int                   no_nodes;             /* Number of nodes */
    int                   ipindex;              /* Location of this node
                                                   in the IPtable */
    int                   current_state;        /* Current state
                                                   (1:Cluster Head.0:Slave)
*/

};


/* Function prototype declarations */

int initST( int argc, char *argv[], ST * );
int initRT( ST *, RT * );
unsigned long int currenttime_sec( void );
int reorder_iptable( ST * );
int locate_myip( int, ST * );
int sendHB( unsigned long int, ST *, RT * );
int procHB( char *, unsigned long int, ST *, RT * );
int client_socket( int );
int server_socket( int );
int analyzeRT( unsigned long int, ST *, RT *, int );
int sendTK( unsigned long int , ST *, RT * );
int processTK( char *, ST *, RT * );
int sendRTmsg( ST *, RT * );
int processRTmsg( char *, unsigned long int , ST *, RT * );
int reorderRT( ST *, RT * );



/*=========================================================================
*/
/* End of File :  proto_header.h
*/
/*=========================================================================
*/
/*=========================================================================
*/
/* Filename      : protocol.c
*/
/* Definition    : Link Establishment Protocol - ANSI C code
*/
```

```
/*=============================================================================
*/

#ifdef SSP_VXW

/* VxWorks header files and defines */

#include <vxWorks.h>
#include <timers.h>
#include <selectLib.h>
#include <sockLib.h>
#include <inetLib.h>
#include <hostLib.h>
#include <time.h>

#define STDIN_FILENO STD_IN

#else

/* Non-VxWorks header files */

#include <sys/time.h>

#endif

/* Common header files */

#include "proto_header.h"
#include <stdio.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <unistd.h>
#include <netdb.h>


#ifdef SSP_VXW

/*
 * VxWorks cannot have a main function.
 * VxWorks cannot have a main() function. proto() replaces the main()
function.
 * test() calls proto() with appropriate arguments. While test() is
functional,
 * it is meant mainly as a example of how proto() might be used.
 */
int   proto( int argc, char *argv[] );

int test( void )
{
```

```
    int   rv;
    int   argc;
    char  *argv[4] =
    {
       "proto",
       "192.168.1.100",
       "192.168.1.101",
       "192.168.1.14"
    };

    argc = 4;
    rv = proto(argc, argv);
    return rv;
}

/* VxWorks main() --> proto() */
int proto( int argc, char *argv[] )

#else

/* Non-VxWorks main() */
int main( int argc, char *argv[] )

#endif
{
    int                 portno[SERVER_PORTS];
    int                 maxDescriptor;
    int                 i;
    int                 jj = 1;
    int                 ssockid[SERVER_PORTS];
    int                 numbytes;
    socklen_t           addrlen;
    int                 running = 1;
    int                 HBtimeout;
    char                msg[MAXBUFSIZE];
    char                senderip[20];
    unsigned long int   nextHBTime;
    unsigned long int   nextTKTime;
    unsigned long int   mySeqNo = 0;
    unsigned long int   timeout = 10;
    unsigned long int   currtime;
    struct  timeval     selTimeout;
    struct  sockaddr_in client_addr;
    fd_set              sockSet;
      ST st;
      RT rt[MAX_NODES];


    /* Check for proper arguments */

    if(argc <=1)
    {
        fprintf(stderr, " Usage: proto argc, IP_ADDR1, IP_ADDR2, ...\n");
        return -1;
    }
```

```
    /* Initialize state table and Routing table */

    initST( argc, argv, &st );    /* Initialize Statetable */
    initRT( &st, &rt[0] );        /* Initialize Routing table */

    /* Client - Server Module */

    portno[0] = HBRX_PORT;
    portno[1] = HSRX_PORT;
    portno[2] = TOKENRX_PORT;

    maxDescriptor = -1;

    /* Create server sockets and store their descriptors */

    for( i =0; i<SERVER_PORTS; i++ )
    {
        ssockid[i] = server_socket(portno[i]);

        if(ssockid[i] > maxDescriptor)
            maxDescriptor = ssockid[i];
    }

    currtime    = currenttime_sec( );
    nextHBTime  = currtime;
    nextTKTime  = currtime;
    HBtimeout     = 3; /* Can be varied. Timeouts on 3 Heartbeat msg losses
*/

    /* Runs until "ENTER" Key pressed */

    while( running )
    {
        printf("HOST: RUN COUNT : %d \n",jj);

        FD_ZERO( &sockSet );
        FD_SET( STDIN_FILENO, &sockSet );

        for( i=0; i<SERVER_PORTS; i++ )
            FD_SET( ssockid[i], &sockSet );

        selTimeout.tv_sec = timeout;
        selTimeout.tv_usec =0;

        /* Client Module: Send packets */

        currtime = currenttime_sec( );

        if( currtime>nextHBTime )
        {
            /* Send a Heartbeat message and change the next HB time */

            mySeqNo = mySeqNo+2;
            printf("HOST : Heartbeat transmitted at %lu seconds\n",currtime);

            sendHB( mySeqNo, &st, &rt[0] );
            nextHBTime += HBINTERVAL;
```

```
        /*
         * Update the sequence number and timer corresponding to
         * this node in the Routing Table
         */

        for( i = 0; i<st.no_nodes; i++ )
        {
            if( st.my_ID == rt[i].ID )
            {
                rt[i].seq_no    = mySeqNo;
                rt[i].timer     = currtime;
            }
        }

    }   /* End to check if HB should be sent or not */

    /* Generate a new Token only if this node is a Cluster Head */

    if( currtime>nextTKTime && st.current_state == 1 )
    {

        /*
         * Send a Token and change next Token time
         * Should check if the st.ipindex[i] has even seqno greater
         * or equal to 2.
         */

        if( rt[1].seq_no >= 2 && rt[1].seq_no %2 == 0 )
        {
            printf("HOST : TOKEN sent at %lu seconds \n",currtime);
            sendTK( st.iptable[1], &st, &rt[0] );
            nextTKTime = nextTKTime+TKINTERVAL;
        }

    }   /* End to check if Token should be sent or not */

    /* Server Module: Listen for packets on all the LISTEN ports */

    if( select( maxDescriptor + 1, &sockSet, NULL, NULL, &selTimeout ) ==
0 )
        printf("SERVER: %lu second timeout\n",timeout);

    if( FD_ISSET(STDIN_FILENO,&sockSet) )
    {
        printf("SERVER: ENTER Key pressed!!!Exiting program.\n");
        getchar();
        running = 0;
    }

    for( i = 0; i<SERVER_PORTS; i++ )
    {
        if( FD_ISSET(ssockid[i],&sockSet) )
        {
            addrlen     = sizeof(struct sockaddr);
            numbytes    = recvfrom(ssockid[i],msg,sizeof(msg),0,
             (struct sockaddr *) &client_addr, &addrlen);
```

```
                if( numbytes < 0 )
                {
                    perror("SERVER :  Receive message failed");
                    return -1;
                }

                msg[numbytes] = '\0';
                printf("SERVER : RX<--%s: %u %d bytes: %s\n",
                    inet_ntoa(client_addr.sin_addr),
                    ntohs(client_addr.sin_port), numbytes, msg);

                strcpy(senderip ,inet_ntoa(client_addr.sin_addr));

                /* Find the type of message that arrived */

                if( portno[i]==5011 )
                {
                    printf("SERVER: Heartbeat msg received on port no. %d\n",
                    portno[i]);
                    procHB( msg, inet_addr(senderip), &st, &rt[0] );
                }

                else if( portno[i]==5001 )
                {
                    printf("SERVER: Handshake msg received on port no. %d\n",
                    portno[i]);
                    processRTmsg( msg, inet_addr(senderip), &st, &rt[0] );
                }

                else if( portno[i]== 5031)
                {
                    printf("SERVER: Token received on port no. %d\n",
                    portno[i]);
                    processTK( msg, &st, &rt[0] );
                }

            }   /* End of if structure */

        }   /* End of for loop */

        jj++;

        currtime = currenttime_sec( );

        analyzeRT( currtime, &st, &rt[0], HBtimeout );

    } /* End of running */

    /* Close Server Sockets */

    for (i =0; i<SERVER_PORTS; i++)
        close(ssockid[i]);


    printf("SERVER: All sockets closed. Server exiting.\n");
    printf("CLIENT: All sockets closed. Client exiting.\n");
```

```
    return(0);
}

/*.... End of Main method ------------------------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>:  Initialize state table
*/
/* Comments  :  This function accepts the the IP addresses of all the nodes
*/
/*              in the network. The IP address of this node is determined
*/
/*              automatically.
*/
/*-------------------------------------------------------------------------
*/

int initST( int argc, char *argv[], ST *st )
{
      char           s[256];

      #ifdef SSP_VXW
      int            h;
      #else
      struct hostent *h;
      struct in_addr *pina;
      #endif

      struct in_addr ina;
      int            i, j, k;
      int            is_dup;

   /* Get my host name */
   if( gethostname(s, sizeof(s)) < 0 )
   {
      perror("CONFIG: Could not determine host name");
      return -1;
   }

   /* Get my IP address using my host name */
   #ifdef SSP_VXW
   if( (h = hostGetByName(s)) == ERROR )
   {
      perror("CONFIG: Could not determine host info");
      return -1;
   }
   #else
   if( (h = gethostbyname(s)) == NULL )
   {
      herror("CONFIG: Could not determine host info");
      return -1;
```

```
}
#endif

#ifdef SSP_VXW
ina.s_addr  = h;
#else
pina        = (struct in_addr *)h->h_addr;
ina.s_addr  = pina->s_addr;
#endif

st->my_ID                   = ina.s_addr;
st->ipindex                 = 0;
st->iptable[st->ipindex]    = ina.s_addr;
printf("CONFIG: Server IP address %s\n", inet_ntoa(ina));

/*
 * Start out as a slave.
 * (currentstate = 1 implies a CH and 0 implies a slave).
 */
st->current_state = 0;

/* Assign IP addresses to iptable in the state table. */
k = 1;

for( i = 1; i < MAX_NODES; i++ )
{
   /* Stop if there are no more command line arguments. */
   if( k >= argc )
      break;

   /* Convert IP address string to network address. */
   if( inet_aton(argv[k++], &ina) == 0 )
      i--;      /* conversion failed */
   else
   {
      /* See if we already have this address in our table. */
      is_dup = 0;
      for( j = 0; j < i; j++ )
         if( st->iptable[j] == ina.s_addr )
         {
            is_dup = 1;
            break;
         }

      if( is_dup )
         i--;  /* duplicate; decrement node counter */
      else
      {
         /* add new, unique address to table */
         st->iptable[i] = ina.s_addr;
         printf("CONFIG: Node %d IP address %s\n", i, inet_ntoa(ina));
         st->no_nodes = i + 1;
      }
   }
}
 reorder_iptable( st ); /* Arrange IP addresses in ascending order */
 locate_myip( argc, st );
```

```
    printf("INIT : My IP index is %d \n",st->ipindex);
    printf("INIT : Initialization of State table successful\n");

    /* Find out if this node is a CH or slave  */

    if( st->ipindex==0 )
    {
        printf("INIT : This node is a Cluster Head.\n");
        st->current_state = 1; /* Implies it is a Cluster Head */
    }
    else
        printf("INIT: This node is a slave.\n");


   return 0;

}

/*..... End of Initialize State Table ------------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>: Initialize Routing Table
*/
/* Comments  : Needs the state table and Routing Table pointers as
*/
/*             arguments
*/
/*-------------------------------------------------------------------------
*/

int initRT( ST *st, RT *rt)
{
    unsigned long int   curr_time;  /* Current time in seconds */
    int                 i;

    printf("INIT : Initializing Routing Table. \n");

    curr_time = currenttime_sec( );

    for ( i =0; i<st->no_nodes; i++ )
    {
      rt[i].ID                = st->iptable[i];
      rt[i].no_hops           = 1;
      rt[i].nextHop           = st->iptable[i];
      rt[i].seq_no            = 0;
      rt[i].flag        = 0;
      rt[i].timer       = curr_time;
    }

    printf("INIT : Initialization of Routing Table successful. \n");
    return 0;
}
```

```
/*.... End of Initialize Routing Table ------------------------------------
*/


/*-------------------------------------------------------------------------
*/
/* <Function>    : To find the current system time
*/
/* Comment       : Returns time in seconds
*/
/*-------------------------------------------------------------------------
*/

unsigned long int currenttime_sec( void )
{
      unsigned long int      seconds;

      #ifdef SSP_VXW

      struct timespec            tv;
      clock_gettime( CLOCK_REALTIME, &tv );

      #else

      struct timeval         tv;          /* Time value */
      gettimeofday( &tv, NULL );  /* Function to get time of day */
      #endif

      seconds = tv.tv_sec;

    return seconds;
}

/*.... End of Find current time -------------------------------------------
*/




/*-------------------------------------------------------------------------
*/
/* <Function>    : Reorder entires in the IP table
*/
/* Comment       : IP Table is reordered in ascending order of the
*/
/*                 IP addresses
*/
/*-------------------------------------------------------------------------
*/

int reorder_iptable( ST *st )
{
    unsigned long int tempip;
    int               i;
    int               j;
```

```
    for ( i = 0; i<st->no_nodes; i++ )
        for ( j = i+1; j<st->no_nodes; j++)
            if( st->iptable[i] > st->iptable[j] )
            {
                tempip = st->iptable[j];
                st->iptable[j] = st->iptable[i];
                st->iptable[i] = tempip;
            }

    return 0;
}

/*.... End of Re-order IP table ------------------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>    : To find the location of ip address in the IP table
*/
/* Comment       : Updates the location - starting with zero index
*/
/*                 Accepts the number of nodes as argument
*/
/*-------------------------------------------------------------------------
*/

int locate_myip(int nodecnt,ST *st)
{
    int   i;

    for ( i =0; i<nodecnt;i++ )
    {
        if(st->iptable[i] == st->my_ID)
        st->ipindex = i;
     }

    return 0;
}

/* ....End of locate my ip ------------------------------------------------
*/


/*-------------------------------------------------------------------------
*/
/* <Function>    : Create and send Heartbeat
*/
/* Comment       : Accepts Routing table, statetable and sequence number
*/
/*                 Converts the Heartbeat message in to string and sends
*/
/*                 to all 1- hop neighbors except itself
*/
/*-------------------------------------------------------------------------
*/
```

```
int sendHB( unsigned long int mySeqNo, ST *st, RT *rt )
{
    int                     csockid;      /* Client socket ID */
    int                     numbytes = 0; /* Number of bytes(chars)sent by
                                             client */
    int                     lenHB;        /* String length of the Heartbeat
                                             message */
    int                     index;        /* Temporary variable */
    char                    HBmsg[256];   /* Heartbeat message */
    char                    sSeqNo[20];   /* Sequence number as string */
    char                    sFlag[20];    /* Roll-over flag as a string */
    char                    sGentime[20]; /* Time of Hearbeat generation string
*/
    unsigned long int       gentime;      /* Time generation of Heartbeat */
    unsigned long int       flag;         /* Roll-over flag */
    struct sockaddr_in      server_addr;  /* Holds server IP address */


    /* Create HB message (string) */

    gentime = currenttime_sec( );
    flag = 0;

    strcpy(HBmsg, HBID);
    sprintf(sGentime, "%lu", gentime);
    strcat(HBmsg, sGentime);
    strcat(HBmsg, MSGBRK);
    sprintf(sSeqNo, "%lu", mySeqNo);
    strcat(HBmsg, sSeqNo);
    strcat(HBmsg, MSGBRK);
    sprintf(sFlag, "%lu", flag);
    strcat(HBmsg, sFlag);
    strcat(HBmsg, MSGBRK);

    lenHB = strlen(HBmsg);


    for( index =0; index <st->no_nodes; index ++ )
    {

        /* Send HB packets to all nodes except itself and are one hop away */

        if( rt[index].ID != st->my_ID  && rt[index].no_hops == 1 )
        {

            csockid = client_socket(HBTX_PORT);

            /* Creating a socket for the server (destination) */

            memset((char *) &server_addr, 0, sizeof(server_addr));
            server_addr.sin_family      = AF_INET;
            server_addr.sin_addr.s_addr = rt[index].ID;
            server_addr.sin_port        = htons(HBRX_PORT);

            numbytes = sendto(csockid, HBmsg, lenHB, 0,
            (struct sockaddr *) &server_addr, sizeof(server_addr));
```

```
            if( numbytes < 0 )
            {
                perror("CLIENT: Sending message failed");
                return -1;
            }

            printf("CLIENT : TX-->%s : %u %d bytes: %s\n",
                inet_ntoa(server_addr.sin_addr),ntohs(server_addr.sin_port),
                numbytes,HBmsg);


            close(csockid); /* Close client socket */

        }
    }
    return 0;
}

/*.... End of Send Heartbeat ------------------------------------------------
*/


/*---------------------------------------------------------------------------
*/
/* <Function>    : Process Heartbeat
*/
/* Comment       : Accepts the received Heartbeat message string and
*/
/*                 converts them to original data type
*/
/*---------------------------------------------------------------------------
*/

int procHB( char *msg, unsigned long int senderip, ST *st, RT *rt )
{

    int                     initval = 1;      /* Ignore the header(E) */
    int                     ii = 0;
    int                     i;
    int                     new_entry = 1;    /* The received IP is new */
    int                     last_char = 0;
    int                     count = 0;
    char                    rxstring[256];    /* Received string */
    unsigned long int       senderSeqNo;
    unsigned long int       senderflag;
    unsigned long int       sendertime;
    unsigned long int       current_seqno;

    while( msg[ii] !='\0' )
    {

        if( *(msg+initval) != *MSGBRK )
        {
            rxstring[last_char] = msg[initval];
            last_char = last_char+1;
        }
```

```
        else
        {
            count = count+1;

            /* Terminate the string and remove the excess chars */

            *(rxstring+last_char) = '\0';
            last_char =0;

            switch ( count )
            {
                case 1:     /* The received string is time generation of HB
*/
                sscanf(rxstring,"%lu",&sendertime);
                break;

                case 2:     /* The received string is sequence number */
                sscanf(rxstring,"%lu",&senderSeqNo);
                break;

                case 3:     /* The received string is Roll-over Flag */
                sscanf(rxstring,"%lu",&senderflag);
                break;

                default:
                printf("PROCHB : Shouldn't get here \n");
                break;
            }

        } /* End of else structure */

        ii++;
        initval++;

    } /* End of while structure */


    /* Updating the Routing Table */

    for( i = 0; i<st->no_nodes; i++ )
    {

        /* To check if the HB sending node has an entry in the RT */

        if( rt[i].ID == senderip )
        {
            new_entry         = 0; /* Make it as an existing IP address */
            current_seqno     = rt[i].seq_no; /* Store the existing seq. no
                                                 before RT update */

            if( rt[i].seq_no < senderSeqNo ) /* If the HB message is new */
            {
                rt[i].seq_no = senderSeqNo;

                if( (current_seqno % 2) != 0 )
                {
```

```
                    /*
                     * This host was unreachable earlier. So send a RT update
                     * to neighbors.Create Client socket and send the
                     * RTU msg
                     */
                    sendRTmsg( st, &rt[0] );
                }

            }

        }

    } /* End of FOR loop */

    if(new_entry == 1)
    {
        /*
         * Add it to the routing table and reorder the routing table,
         • update state table and IP table. If index of this node
         * is zero in Routing Table change the state to Cluster Head
         */

        rt[st->no_nodes].ID = senderip;
        rt[st->no_nodes].no_hops = 1;
        rt[st->no_nodes].nextHop = senderip;
        rt[st->no_nodes].seq_no = senderSeqNo;
        rt[st->no_nodes].flag = senderflag;
        rt[st->no_nodes].timer = sendertime;

        /* Increment the number of nodes in the statetable */

        st->iptable[st->no_nodes] = senderip;
        st->no_nodes = st->no_nodes+1;
        reorder_iptable( st );

        locate_myip( st->no_nodes, st ); /* Find location my_ID in the IP
table */
        reorderRT( st, &rt[0] ); /* Reorder the Routing Table in Ascending
order */

        if(rt[0].ID == st->my_ID)
        {
            st->current_state = 1;
            printf("This node is a Cluster Head.\n");
        }
        else
        {
            st->current_state = 0;
            printf("This node is a Slave.\n");
        }

        /* Send the updated RT to all neighbors */

        sendRTmsg( st, &rt[0] );

    }
```

```
    return 0;
}

/*.... End of Process Heartbeat ------------------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>    : Create and bind to client socket
*/
/* Comment       : Returns a client socket id upon successful creation
*/
/*                 of a socket
*/
/*-------------------------------------------------------------------------
*/

int client_socket( int portno )
{
    int                 sockid;
    struct sockaddr_in  my_addr;

    /* Creating a Client socket */

    sockid = socket(AF_INET, SOCK_DGRAM, 0);

    if ( sockid < 0 )
    {
      perror("CLIENT : Socket creation failure.");
      return -1;
    }

    /* Binding a socket with port number on the local machine */

    memset((char *) &my_addr, 0, sizeof(my_addr)); /* Write zeros to byte
string */
    my_addr.sin_family      = AF_INET;
    my_addr.sin_port        = htons(portno);
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY); /* automatically fills IP
                                                    of the sending node */

    if ( (bind(sockid,(struct sockaddr *) &my_addr, sizeof(my_addr)) < 0) )
    {
        perror("CLIENT: Socket bind failure.");
        return -1;
    }

    return sockid;
}

/*.... End of create client socket ---------------------------------------
*/
```

```
/*-------------------------------------------------------------------------
*/
/* <Function>    : Create and bind a server socket
*/
/* Comment       : Accepts a listening port number and returns
*/
/*                 the server socket id upon successful creation
*/
/*-------------------------------------------------------------------------
*/

int server_socket( int portno )
{
  int                    sockid;
  struct sockaddr_in     my_addr;

  /*  Creating a socket  */

  printf("SERVER : Creating a socket for port no. %d\n",portno);
  sockid = socket(AF_INET, SOCK_DGRAM, 0);

  if (sockid < 0)
    {
      perror("SERVER: Socket creation failure.");
      return -1;
    }


  /* Binding a socket to a local port  */

  printf("SERVER : Binding socket to port no. %d\n",portno);

  memset((char *) &my_addr, 0, sizeof(my_addr));
  my_addr.sin_family         = AF_INET;
  my_addr.sin_addr.s_addr    = htonl(INADDR_ANY); /* IP address of this node
*/
  my_addr.sin_port           = htons(portno);

  if( ( bind(sockid, (struct sockaddr *) &my_addr, sizeof(my_addr) ) < 0) )
  {
    perror("SERVER : Socket bind failure");
    return -1;
  }

  return sockid;

}

/*..... End of Create and bind a server socket ------------------------------
*/
```

```
/*-------------------------------------------------------------------------
*/
/* <Function>    : Analyze Routing Table
*/
/* Comment       : Accepts current time in seconds
*/
/*                 Analyzes the routing table based on the Heartbeat message
*/
/*                 received at the end of each RUN and if significant change
*/
/*                 has occured, sends out a Routing Table update message.
*/
/*-------------------------------------------------------------------------
*/

int analyzeRT( unsigned long int curr_time , ST *st, RT *rt, int tout )
{
    int                     i;
    int                     send_update = 0;
    unsigned long int       time_difference;


    for( i = 0; i<st->no_nodes; i++ )
    {
        if(curr_time > rt[i].timer)
            time_difference = curr_time - rt[i].timer;

        else
            time_difference = rt[i].timer - curr_time;

        if( time_difference > tout*HBINTERVAL )
        {
            /*
             * Heartbeat timeout for rt[i].ID
             */

            if (rt[i].seq_no % 2 == 0) /* Checking if it was even before */
                rt[i].seq_no = rt[i].seq_no + 1;

            else
                rt[i].seq_no = rt[i].seq_no + 2; /* odd+2 = odd */

            rt[i].no_hops = 1000;
            send_update = 1;
        }
    }

    /* rrorder RT with ODD SEQ/hop count = 1000 NO at BOTTOM */
    reorderRT(st,rt);

    /* Send a Routing Table update message */

    if (send_update == 1)
    {
        sendRTmsg( st, &rt[0] );
    }
```

```
    /*
        Module to check transitions - Cluster Head to slave or
        viceversa
    */

    if(rt[0].ID == st->my_ID)
        st->current_state = 1;
    else
        st->current_state = 0;

    return 0;
}

/* ....End of Analyze Routing Table ---------------------------------------
*/




/*-------------------------------------------------------------------------
*/
/* <Function>    : Create and send Token
*/
/* Comment       : Accepts destination IP address
*/
/*                 Converts the Token message in to string and sends
*/
/*                 to destination
*/
/*-------------------------------------------------------------------------
*/


int sendTK( unsigned long int destination, ST *st, RT *rt )
{
    int                   csockid;          /* Client socket ID */
    int                   numbytes;         /* Number of bytes(chars)sent */
    int                   lenTK;            /* String length of the msg */
    int                   i;                /* Temporary variable */
    char                  TKmsg[256];       /* Token message string */
    char                  sGentime[20];     /* String for Token generation */
    char                  sExptime[20];     /* String for Token Expiry time
*/
    char                  sDestination[20]; /* Destination IP as string */
    unsigned long int     next_dest = 0;    /* Next destination IP */
    unsigned long int     gentime;          /* Time at generation of Token */
    unsigned long int     expirytime;       /* Time to live for Token */
    struct sockaddr_in    server_addr;      /*  Stores server IP address */

    gentime = currenttime_sec( );
    expirytime = 5*gentime;

    /* Convert to string */
    sprintf(sGentime,"%lu",gentime);
    sprintf(sExptime,"%lu",expirytime);
    sprintf(sDestination,"%lu",destination);
```

```
    strcpy(TKmsg,sGentime);
    strcat(TKmsg,MSGBRK);
    strcat(TKmsg,sExptime);
    strcat(TKmsg,MSGBRK);
    strcat(TKmsg,sDestination);
    strcat(TKmsg,MSGBRK);

    /* Create Token message */

    lenTK = strlen(TKmsg);

    csockid = client_socket( TOKENTX_PORT );

    /*
     *  next_dest can be either the final destination or next hop to final
     *  destination.Find this from the routing table.
     */

    for( i = 0; i<st->no_nodes; i++ )
    {
        if( destination == rt[i].ID )
        {
            if( rt[i].no_hops == 1 )
                next_dest = rt[i].ID;
            else
                next_dest = rt[i].nextHop;
        }
    }


    /* Creating a socket for the server (destination) */

    memset((char *) &server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = next_dest;
    server_addr.sin_port = htons(TOKENRX_PORT);


    numbytes = sendto(csockid,TKmsg,lenTK,0,(struct sockaddr *) &server_addr,
                sizeof(server_addr));

    if (numbytes < 0)
    {
        perror("CLIENT: Sending TOKEN message failed");
        return -1;
    }

    printf("CLIENT : TX-->%s : %u %d bytes: %s\n",
        inet_ntoa(server_addr.sin_addr),ntohs(server_addr.sin_port),
        numbytes,TKmsg);

    close(csockid); /* Close client Socket */

    return 0;
}
```

```
/*.....End of Create and send Token message --------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>    : Process Token
*/
/* Comment       : Accepts Token message as a string.
*/
/*                 Processes the Token and passes it on to the next node in
*/
/*                 the IP table OR forwards it
*/
/*-------------------------------------------------------------------------
*/

int processTK( char *Token, ST *st , RT *rt )
{
    int                     count = 0;
    int                     ii = 0;
    int                     initval = 0;
    int                     last_char = 0;
    char                    rxstring[256];
    unsigned long int       expiry_time;
    unsigned long int       generation_time;
    unsigned long int       cur_time;
    unsigned long int       destinationip;
    unsigned long int       next_dest;


    /* Convert string to original data types */

    while( Token[ii] != '\0')
    {

        if( *(Token+initval) != *MSGBRK )
        {
            rxstring[last_char] = Token[initval];
            last_char = last_char + 1;
        }
        else
        {
            count= count + 1;

            *(rxstring+last_char) = '\0';
            last_char = 0;

            switch( count )
            {

                case 1:     /* Means an unsigned long int */
                sscanf(rxstring,"%lu",&generation_time);
                break;
```

```
                case 2:     /* Means an unsigned long int */
                sscanf(rxstring,"%lu",&expiry_time);
                break;

                case 3:     /* Means an unsigned long int */
                sscanf(rxstring,"%lu",&destinationip);
                break;

                default:
                printf("Shouldn't get here \n");
                break;

            }

        } /* End of else */

        ii++;
        initval++;

    } /* End of while */


    /* Check if the expiry time has occured ???*/

    cur_time = currenttime_sec( );

    if(cur_time > expiry_time)
    {
        /* Token time has elapsed */

        printf("TKPROC : The time to live for Token has expired. \n");
        printf("TKPROC : Discarding Token ....\n"); /* Do nothing */

    }

    else
    {
        /*
         *  Check if Token is destined for itself. If a token is destined
         *  and reaches the Cluster Head, the Token should be discarded
         *  after data transfer.
         */

        if( st->my_ID == destinationip )
        {

            /* Token is for this host */

            printf("TKPROC: This TOKEN is for THIS Node.Send DATA ......\n");

            /*
             *  After data transfer either forward Token to the next node
             *  in the iptable OR send it to the Cluster Head. This would
             *  happen if only if this node is not the CH.
             */
```

```
            if( st->current_state == 0 )
            {
                if(st->ipindex == st->no_nodes-1)
                next_dest = st->iptable[0]; /* Send to CH */

                else
                next_dest = st->iptable[st->ipindex+1]; /* Send to Next node
*/

                sendTK( next_dest , st, &rt[0] );
            }

        }

    }  /* End of else (process Token) */

    return 0;
}

/*.....End of Process Token ------------------------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>    : Create and send Routing table message
*/
/* Comment       : Converts Routing Table message in to a string , creates a
*/
/*                 client socket and sends it to the destination.
*/
/*-------------------------------------------------------------------------
*/

int sendRTmsg( ST *st, RT *rt )
{
    int                 csockid;      /* Client socket ID */
    int                 numbytes;     /* Number of bytes(chars)sent */
    int                 lenRTmsg;     /* String length of the RT */
    int                 i;            /* Temporary variable */
    char                RTmsg[1024];  /* Routing Table message */
    char                sID[20];      /* String value of destination */
    char                sNohops[20];
    char                sNextHop[20];
    char                sSeqNo[20];
    char                sFlag[20];
    char                sTimer[20];
    struct sockaddr_in  server_addr;  /*  Stores server IP address */


     /* Create RT message string */

    strcpy (RTmsg,RTTBL);

    for( i=0; i<st->no_nodes; i++ )
```

```
{

    /* Convert non-string data types to string */
    sprintf(sID, "%lu", rt[i].ID);
    sprintf(sNohops, "%d", rt[i].no_hops);
    sprintf(sNextHop, "%lu", rt[i].nextHop);
    sprintf(sSeqNo, "%lu", rt[i].seq_no);
    sprintf(sFlag, "%d", rt[i].flag);
    sprintf(sTimer, "%lu", rt[i].timer);

    /* Concatenate to form a message string */

    strcat(RTmsg,sID);
    strcat(RTmsg,MSGBRK);
    strcat(RTmsg,sNohops);
    strcat(RTmsg,MSGBRK);
    strcat(RTmsg,sNextHop);
    strcat(RTmsg,MSGBRK);
    strcat(RTmsg,sSeqNo);
    strcat(RTmsg,MSGBRK);
    strcat(RTmsg,sFlag);
    strcat(RTmsg,MSGBRK);
    strcat(RTmsg,sTimer);
    strcat(RTmsg,MSGBRK);

}


lenRTmsg = strlen(RTmsg);

for( i=0; i<st->no_nodes; i++ )
{
    /* Check if the destination is reachable */

    if(rt[i].seq_no % 2 == 0)
    {
        /*
         *  Send RT packets to all nodes except itself and are
         *  one hop away
         */

        if( (rt[i].ID != st->my_ID) && rt[i].no_hops ==1 )
        {
            csockid = client_socket( HSTX_PORT );

            /* Creating a socket for the server (destination) */

            memset((char *) &server_addr, 0, sizeof(server_addr));
            server_addr.sin_family = AF_INET;
            server_addr.sin_addr.s_addr = rt[i].ID;
            server_addr.sin_port = htons(HSRX_PORT);

            numbytes = sendto(csockid,RTmsg,lenRTmsg,0,
             (struct sockaddr *) &server_addr,sizeof(server_addr));

            if (numbytes < 0)
            {
```

```
                    perror("CLIENT: Sending Routing Table failed");
                    return -1;
                }

                close(csockid); /* Close client Socket */
            }
        }
    }

    return 0;
}

/*.... End of create and send Routing Table --------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>    : Process Routing table message
*/
/* Comment       : Accepts the Routing Table message as a string
*/
/*                 and converts them back to their original data type
*/
/*-------------------------------------------------------------------------
*/

int processRTmsg( char *RRTmsg, unsigned long int sourceip, ST *st, RT *rt )
{
    int      initval = 1;
    int      last_char = 0;
    int      ii=0;
    int      count = 0;
    int      i = 0;
    int      entry_found;
    int      send_update = 0;
    int      j;
    int      no_new_nodes = 0; /* Number of new nodes */
    int      rx_no_nodes = 0; /* Number of nodes in the received RT */
    int      old_no_nodes = st->no_nodes; /* Number of nodes in the existing
RT */
    char     rxstring[20];
    RT       newRT[MAX_NODES];

    /*
     * Separate strings by navigating through the entire string and
     * breaking in to substrings. The init value is 1 higher to ignore
     * identifier "D"
     */

    while( RRTmsg[ii] !='\0' )
    {

        if( *(RRTmsg+initval) != *MSGBRK )
        {
```

```
            rxstring[last_char] = RRTmsg[initval];
            last_char = last_char+1;
        }

        else
        {
            count = count +1;

            /* To terminate the string and remove the excess chars */

            *(rxstring+last_char) = '\0';
            last_char =0;

            switch ( count )
            {
                case 1:
                sscanf( rxstring, "%lu", &newRT[i].ID );
                rx_no_nodes++;
                break;

                case 2:
                sscanf( rxstring, "%d", &newRT[i].no_hops );
                break;

                case 3:
                sscanf( rxstring, "%lu",  &newRT[i].nextHop );
                break;

                case 4:
                sscanf( rxstring, "%lu", &newRT[i].seq_no );
                break;

                case 5:
                sscanf( rxstring, "%d", &newRT[i].flag );
                break;

                case 6:
                sscanf( rxstring, "%lu", &newRT[i].timer );
                break;

                default:
                printf("Shouldn't get here \n");
                break;
            }

            if ( count == 6 )
            {
                count = count - 6;
                i++;
            }
        }   /* End of else */

        ii++;
        initval++;

    }   /* End of while */
```

```
/*
 * Compare the received routing table with existing routing table.
 * Change values in the existing routing table and also the state
 * table. Send the update to all neigbors
 */

for( i = 0; i<rx_no_nodes; i++ )
{

    entry_found = 0;

    for ( j = 0; j<st->no_nodes; j++ )
    {

        /* IP address already in the Routing Table */

        if( newRT[i].ID == rt[j].ID )
        {
            entry_found = 1;

            /* Received seqeunce number is latest */

            if( rt[j].seq_no < newRT[i].seq_no )
            {
                /*
                 * Existing seq. number is odd(unreachable host)
                 * received seq. number is even(reachable host)
                 */

                if( rt[j].seq_no %2 != 0 && newRT[i].seq_no %2 == 0 )
                    send_update = 1;

                /* Lesser number of hops */

                else if( rt[j].no_hops > newRT[j].no_hops+1 )
                    send_update = 1;

                rt[j].nextHop   = sourceip;
                rt[j].no_hops   = newRT[i].no_hops + 1;
                rt[j].seq_no    = newRT[i].seq_no;
                rt[j].flag      = newRT[i].flag;
                rt[j].timer     = newRT[i].timer; /* Store the timer ? */
            }

        }

    } /* End of FOR loop j */

    if( entry_found == 0 ) /* New node */
    {
        rt[old_no_nodes].ID         = sourceip;
        rt[old_no_nodes].nextHop    = sourceip;
        rt[old_no_nodes].no_hops    = newRT[i].no_hops + 1;
        rt[old_no_nodes].seq_no     = newRT[i].seq_no;
        rt[old_no_nodes].flag       = newRT[i].flag;
        rt[old_no_nodes].timer      = newRT[i].timer; /* Store the timer
*/
```

```
            st->iptable[old_no_nodes]     = sourceip; /* Add node in IP table
*/
            old_no_nodes++;
            no_new_nodes++;

            send_update = 1;

        }

    } /* End of FOR loop i */


    /*
     * Update state table with the new number of nodes and reorder the
Routing
     * table. Check if a transition from Cluster Head to slave or viceversa
     * should occur based on the reordering.
     */

    st->no_nodes = st->no_nodes + no_new_nodes;
    reorder_iptable( st );
    reorderRT( st, &rt[0] );

    if( st->iptable[0] == st->my_ID )
        st->current_state = 1; /* This node is CH */

    else
        st->current_state = 0; /* This node is a slave */

    if( send_update == 1 )
    {
        /* Send the New Routing table to all nodes */
        sendRTmsg( st, &rt[0] );
    }

    return 0;

}

/*.... End of Process Routing Message ------------------------------------
*/



/*-------------------------------------------------------------------------
*/
/* <Function>    : Reorder entires in the Routing Table
*/
/* Comment       : Routing Table is reordered in ascending order of the
*/
/*                 IP addresses
*/
/*-------------------------------------------------------------------------
*/
```

```
int reorderRT( ST *st, RT *rt )
{

    RT    temprt;  /* Temporary RT variable */
    int   i;
    int   j;
    int   ii;
    int   unreachable_index[MAX_NODES];


    for (i = 0; i<st->no_nodes; i++)

        for (j = i+1; j<st->no_nodes; j++)

            if ( rt[i].ID > rt[j].ID )
            {

                temprt.ID       = rt[j].ID;
                temprt.no_hops  = rt[j].no_hops;
                temprt.nextHop  = rt[j].nextHop;
                temprt.seq_no   = rt[j].seq_no;
                temprt.flag     = rt[j].flag;
                temprt.timer    = rt[j].timer;

                rt[j].ID        = rt[i].ID;
                rt[j].no_hops   = rt[i].no_hops;
                rt[j].nextHop   = rt[i].nextHop;
                rt[j].seq_no    = rt[i].seq_no;
                rt[j].flag      = rt[i].flag;
                rt[j].timer     = rt[i].timer;

                rt[i].ID        = temprt.ID;
                rt[i].no_hops   = temprt.no_hops;
                rt[i].nextHop   = temprt.nextHop;
                rt[i].seq_no    = temprt.seq_no;
                rt[i].flag      = temprt.flag;
                rt[i].timer     = temprt.timer;

            }

            /*
             * All the unreachable nodes should be brought to the end of the
             * routing table.
             */

            j = 0;

            for ( i = 0; i<st->no_nodes; i++ )
            {
                if (rt[i].no_hops == 1000)
                {
                    unreachable_index[j] = i;
                    j++;
                }
            }

            if( j > 0 )
```

```
            {
                for ( i = 0; i<j; i++ )
                {
                    for ( ii = unreachable_index[i]; ii<st->no_nodes; ii++ )
                    {
                        temprt.ID       = rt[ii].ID;
                        temprt.no_hops  = rt[ii].no_hops;
                        temprt.nextHop  = rt[ii].nextHop;
                        temprt.seq_no   = rt[ii].seq_no;
                        temprt.flag     = rt[ii].flag;
                        temprt.timer    = rt[ii].timer;

                        rt[ii].ID        = rt[ii+1].ID;
                        rt[ii].no_hops   = rt[ii+1].no_hops;
                        rt[ii].nextHop   = rt[ii+1].nextHop;
                        rt[ii].seq_no    = rt[ii+1].seq_no;
                        rt[ii].flag      = rt[ii+1].flag;
                        rt[ii].timer     = rt[ii+1].timer;

                        rt[ii+1].ID        = temprt.ID;
                        rt[ii+1].no_hops   = temprt.no_hops;
                        rt[ii+1].nextHop   = temprt.nextHop;
                        rt[ii+1].seq_no    = temprt.seq_no;
                        rt[ii+1].flag      = temprt.flag;
                        rt[ii+1].timer     = temprt.timer;

                    }

                    unreachable_index[i+1] = unreachable_index[i+1]-1;
                }
            }

    return 0;
}

/*.... End of Re-order routing table ------------------------------------
*/
```

Appendix B – Autoconfig Receiver Code

```
/*************************************************************************/
/* Filename : readme.txt                                                 */
/* Purpose  : This file gives the procedure for testing the              */
/*            data rate estimation software                              */
/*************************************************************************/

This software release contains one C file, one C MEX-file, one header
file and two matlab files. The two matlab files (analog_sim.m and
modulate.m) and one C MEX-file (mex_data_rate_est.c) are used only for
generating data for testing the algorithm. The estimation algorithm
consists of one C file (data_rate_estimate.c) and one header file
(data_rate_estimate.h). The data_rate_estimate.c contains est_data_rate(),
which is the main function that performs the estimation.

Test procedure using MATLAB test code
=====================================
The following is the test procedure using MATLAB version 6.5:
1. Set the directory in MATLAB to the path of the directory containing the
   release code.
2. Compile the C files on MATLAB using the following command in the
   command prompt of MATLAB.

   >>mex mex_data_rate_est.c data_rate_estimate.c

3. The main test file is the analog_sim.m. It generates data and calls
   the data rate estimation algorithm. The usage for analog_sim can be
   obtained by typing 'help analog_sim' and the usage is as shown below:

     ANALOG_SIM simulator for testing the data rate estimation algorithm.

     ANALOG_SIM(REPS) tests the algorithm that estimates the data rate of a
     binary transmitted sequence. It is executed for REPS number of
     repetitions.

    E.g. for usage of analog_sim

    >>analog_sim(10)
```

```
function analog_sim(reps);
% ANALOG_SIM simulator for testing the data rate estimation algorithm.
%
%   ANALOG_SIM(REPS) tests the algorithm that estimates the data rate of a
%   binary transmitted sequence. It is executed for REPS number of
%   repetitions.

% sample rate
global sample_rate;

sample_rate = 3 * 10^6; % sample rate in samples/s

% List of data rates in bits per second
data_rate = ...
[1000
 50000
 75000
 100000
 125000
 150000
 175000
 200000
 225000
 250000
 275000
 300000
];

signal_time_len = 1.0;  % Duration of data in seconds
data_rate_len = length(data_rate); % Number of data rates

Eb_No_dB_list = [17 16 14 12 10 8 5 4 2]; % List of Eb/No in dB

% Number of Eb/No values for which the estimation algorithm is tested for.
NUM_EB_NO_VALUES = length(Eb_No_dB_list);

%-------------------
% Initialize arrays
%-------------------
% Buffer containing data rate error
rate_err_nrz = zeros(reps, data_rate_len);

% Buffer containing the relative data rate error
nrz_rel_error = zeros(data_rate_len, NUM_EB_NO_VALUES);

% Signal to noise ratio
SNR = zeros(data_rate_len, NUM_EB_NO_VALUES);

data_format = 'NRZ'; % Data format of the modulated data

% Testing is carried out for Eb/No values in the range 17 dB  to 2 dB
for snr_count = 1:length(Eb_No_dB_list)
    Eb_No_dB = Eb_No_dB_list(snr_count);
    for n = 1 : data_rate_len
        number_of_bits = ceil(data_rate(n) * signal_time_len); % Number of bits
        Eb_No = 10^(Eb_No_dB/10);
```

```
        % Noise variance
        noise_var = sample_rate/ (Eb_No * data_rate(n));

        % SNR
        SNR(n,snr_count) = 1/noise_var;

        % Standard deviation of the noise signal
        std_dev = sqrt(noise_var);

        for m = 1 : reps
            data_seq = randint(1, number_of_bits);

            % Modulate the data
            sig_waveform = modulate(data_seq, data_rate(n), data_format);

            % generate noise
            randnoise = std_dev * randn(1,length(sig_waveform));

            % Add noise
            sig_waveform = sig_waveform + randnoise;

            % Data rate estimation algorithm
            est_rate = mex_data_rate_est(sig_waveform);

            % Squared error of the data rate
            rate_err_nrz(m,n) = (est_rate - data_rate(n))^2;
        end % for m = 1 : reps

        % Relative data rate error in percentage
        nrz_rel_error(n,snr_count) = ...
            sqrt(mean(rate_err_nrz(:,n)))/data_rate(n) * 100

    end % for n = 1 : data_rate_len
end % snr_count = 1:length(Eb_No_dB_list)
%-------------------------------------------------------------------------
% End of File
%-------------------------------------------------------------------------


function outData = modulate(inData, data_rate_bps, data_format)

% MODULATE modulates the data to the given data rate and format
%
%   OUTDATA = MODULATE(INDATA, DATA_RATE_BPS, DATA_FORMAT) modulates the
%   input binary data INDATA to a given data rate DATA_RATE_BPS in bits
%   per second and a given data format DATA_FORMAT (viz. either NRZ or
%   Biphase) and returns the modulated data OUTDATA.

global sample_rate;

% data sequence
data_sequence = inData;

% length of the data sequence
length_data_seq = length(data_sequence);

if(strcmp(data_format,'NRZ'))
```

```
    count = 1;

    % determine index at which data transition occurs
    diff = (data_sequence ~= circshift(data_sequence,[0,1]));
    diff = diff(2:end); % The first bit is not required
    index = find(diff);

    bit_data = data_sequence(index);

    clear data_sequence;

    % Number of transition
    num_transitions = length(index);

    % Determine the data length upto transition
    data_length = round(index * (sample_rate / data_rate_bps));

    start_index = data_length + 1;
    start_index = [1 start_index];

    total_length = data_length(end);
    outData = zeros(1,total_length);

    % A MEX file was written for the following "for-loop". Since the
    % MEX file didnot improve the execution time of the software, the
    % MATLAB code is retained.
    for k = 1 : (num_transitions);
        outData(start_index(k) : data_length(k)) = -(-1).^bit_data(k);
    end
else % Bi-phase
    count = 1;

    % Generate a data sequence that is equivalent to Bi-phase
    new_data_seq = zeros(1,2 * length_data_seq);

    % Convert 0 to [0 1] and 1 to [1 0]
    data_sequence_temp = data_sequence + 1;
    data_sequence_bin = dec2bin(data_sequence_temp);

    clear data_sequence_temp;
    clear data_sequence;

    count = 1;
    for k = 1:size(data_sequence_bin)
        % ASCII value of 0 is 48
        new_data_seq(count : count + 1) = data_sequence_bin(k,:) - 48;
        count = count + 2;
    end

    length_data_seq = count - 2;

    clear data_sequence_bin;

    count = 1;
    % determine index at which data transition occurs
    diff = (new_data_seq ~= circshift(new_data_seq,[0,1]));
    diff = diff(2:end); % The first bit is not required
```

```
    index = find(diff);
    bit_data = new_data_seq(index);

    % Number of transition
    num_transitions = length(index);

    clear new_data_seq

    % Determine the data length upto transition
    data_length = round(index * (sample_rate / (2 * data_rate_bps)));

    start_index = data_length + 1;
    start_index = [1 start_index];
    total_length = data_length(end);
    outData = zeros(1,total_length);

    % A MEX file was written for the following "for-loop". Since the
    % MEX file didnot improve the execution time of the software, the
    % MATLAB code is retained.
    for k = 1 : (num_transitions);
        outData(start_index(k) : data_length(k)) = -(-1).^bit_data(k);
    end
end

clear start_index
clear bit_data
return;
%-------------------------------------------------------------------------
% End of file
%-------------------------------------------------------------------------


/*****************************************************************************/
/* File name : mex_data_rate_est.c                                           */
/*                                                                           */
/* Purpose: This file contains the functions mexFunction() and               */
/*          test_est_datarate(). mexFunction() serves as an interface        */
/*          between the MATLAB code that generates the data and the          */
/*          data rate estimation algorithm that is in ANSI C. The            */
/*          test_est_datarate() calls the data rate estimation algorithm.    */
/*****************************************************************************/

/* List of included header files */
#include <stdio.h>
#include <stdlib.h>
#include "mex.h"
#include "data_rate_estimate.h"

/* List of Macros */
#define SAMPLE_RATE   (3000000) /* 3 Million samples per second */

/*===========================================================================*/
/* Function name : test_est_datarate()                                       */
/*                                                                           */
/* Input         : txdata: pointer to the buffer containing the transmitted  */
/*                   data.                                                   */
/*                 txdata_len: length of the buffer containing the           */
```

```
/*                  transmitted data.                                        */
/*                 est_rate: pointer to the estimated rate                   */
/*                                                                           */
/* Output        : The function returns 0 on success and -1 on failure       */
/*                                                                           */
/* Purpose       : This function initializes the structures required by the  */
/*                 data rate estimation algorithm and calls the estimation   */
/*                 algorithm.                                                */
/*===========================================================================*/
int test_est_datarate (float *txdata, int txdata_len, double *est_rate)
{
    /* Allocate memory to the various data structures */
     acfgrx_params_t *params_ptr; /* Autoconfig parameter structure pointer */

     /* Allocate memory to sampled data structure */
    sample_data_t *sample_data_ptr = (sample_data_t*)mxMalloc(sizeof(\
        sample_data_t));

    if(sample_data_ptr == NULL)
    {
        mexPrintf("Unable to allocate memory to sample_data structure");
        return(-1);
    }

    /* Allocate memory to the structure containing estimation parameters */
    params_ptr = (acfgrx_params_t *)mxMalloc(sizeof(\
        acfgrx_params_t));

    if(params_ptr == NULL)
    {
        mexPrintf("Unable to allocate memory to acfgrx_params structure \n");

        /* De-allocate previously allocated memory */
        mxFree(sample_data_ptr);
        return(-1);
    }

    /* initialize the sample_data structure elements */
    sample_data_ptr->len = txdata_len;
    sample_data_ptr->rate = SAMPLE_RATE;
    sample_data_ptr->data = (float*)txdata;

    if(est_data_rate(sample_data_ptr, params_ptr) < 0)
    {
        /* De-allocate previously allocated memory */
        mxFree(sample_data_ptr);
        mxFree(params_ptr);
        return(-1);
    }

    *est_rate = (double)params_ptr->data_rate;

    /* De-allocate previously allocated memory */
    mxFree(sample_data_ptr);
    mxFree(params_ptr);

    return(0);
```

```
}

/* mexFunction definition */
/*=============================================================================*/
/* Function name : mexFunction()                                               */
/* Input         :                                                             */
/*                 number of left hand side arguments,                         */
/*                 pointer to array of Left hand side arguments                */
/*                 number of right hand side arguments                         */
/*                 pointer to array of Right hand side arguments               */
/*                                                                             */
/* Output        :                                                             */
/* Returns       : None                                                        */
/*=============================================================================*/
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    /* Pointer to estimated data rate */
    double *est_rate;

    /* Pointer to the transmitted data */
    double *txdata;

    int i;

    float *tx_data_sp; /* Single Precision transmitted data */

    /* total number of elements present in the transmitted data */
    int txdata_len;

    /* Check for the number of input arguments */
    if(nrhs != 1)
    {
        mexErrMsgTxt("Number of input argument should be one");
    }

    /* Check for the array containing the noisy transmitted data(RHS) */
    if( (mxGetN(prhs[0]) == 0) || (mxGetM (prhs[0]) < 1) ||
    !mxIsDouble(prhs[0]) || mxIsComplex (prhs[0]))
    {
        mexErrMsgTxt("txdata must be a real vector");
    }

    /* txdata must be a row vector */
    if (mxGetM(prhs[0]) != 1)
    {
       mexErrMsgTxt("transmitted data must be a row vector.");
    }

    /* Initialize the txdata pointer */
    txdata = mxGetPr(prhs[0]);

    txdata_len = mxGetN(prhs[0]);

    tx_data_sp = (float *)mxMalloc(sizeof(float) * txdata_len);

    if (tx_data_sp == NULL)
    {
```

```
        mexPrintf("Could not allocate memory to tx_data_sp \n");
        return;
    }

    /* Array that contains the estimated data rate */
    plhs[0] = mxCreateDoubleMatrix(1, 1, mxREAL);

    /* Pointer pointed to their respective buffer */
    est_rate = mxGetPr(plhs[0]);

    /* Convert txdata to 32 bit floating point number */
    for (i=0;i < txdata_len; i++)
    {
        tx_data_sp[i] = (float) txdata[i];
    }

    /* Call the routine that initializes the data structures and calls the */
    /* data rate estimation algorithm.                                      */
    test_est_datarate(tx_data_sp, txdata_len, est_rate);

    mxFree(tx_data_sp);
}
/*****************************************************************************/
/* End of File                                                               */
/*****************************************************************************/


/*****************************************************************************/
/* File name : data_rate_estimate.h                                          */
/*                                                                           */
/* Purpose: This header file contains the prototype of the function that     */
/*          estimates the data rate and contains the definition of structures*/
/*          that are used by the estimation algorithm.                       */
/*****************************************************************************/

/***********************************/
/* structure containing sample data */
/***********************************/
struct sample_data {
   int      len;   /* length of the sampled data */
   float    rate;  /* sample rate */
   float    *data; /* sampled data */
};
typedef struct sample_data sample_data_t;

/************************************************/
/* structure containing the estimated parameters */
/************************************************/
struct acfgrx_params {
   float             data_rate;   /* data or baud rate */
};
typedef struct acfgrx_params acfgrx_params_t;

/* List of function prototypes */
/*****************************************************************************/
/* Function name   : est_data_rate                                           */
/* Input arguments : sample_data_ptr: pointer to the sample_data structure   */
```

```
/*                  params_ptr: pointer to the acfgrx_params structure      */
/*                                                                          */
/* Output arguments: The function returns "0" for success and "-1" for      */
/*                      failure.                                            */
/* Purpose         : This is the function that does the estimation of data  */
/*                   rate.                                                  */
/****************************************************************************/
int est_data_rate(sample_data_t *sample_data_ptr, acfgrx_params_t *params_ptr);

/****************************************************************************/
/* End of File                                                              */
/****************************************************************************/


/****************************************************************************/
/* File name : data_rate_estimate.c                                         */
/*                                                                          */
/* Purpose: This file contains the functions that are used in estimating the */
/*          data rate. The functions includes est_data_rate() that estimates */
/*          the data rate, filter_data() that filters a given data using    */
/*          moving average filter, transit_detect() that determines the     */
/*          samples at which transition occurs and hist_fn() that obtains   */
/*          the histogram of the pulse widths.                              */
/*                                                                          */
/****************************************************************************/

/* Included header files */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "data_rate_estimate.h"

/* MATLAB_MEX_FILE macro is defined when MATLAB test interface is used */
#ifdef MATLAB_MEX_FILE
    /* Include the mex header */
    #include "mex.h"

    /* Redefine some of the ANSI-C specific functions to MEX specific */
    /* subroutines */
    #define malloc(x) mxMalloc(x)
    #define calloc(x,y) mxCalloc(x,y)
    #define free(x) mxFree(x)
    #define printf mexPrintf
#endif

/*************************/
/* List of Macros        */
/*************************/
#define MIN_DATA_RATE (1000)     /* 1000 bps */
#define MAX_NUM_BINS  (500)      /* Maximum number of histogram bins */

/* Number of samples for determining occurance of transition */
#define THRESHOLD     (10)

/* This state indicates that no transition has occured */
#define STABLE        (0)
```

```
/* This state indicates that transition may have occured */
#define TENTATIVE     (1)

/* Operation that determines the sign change */
#define sign_change(x,y) ((x) >= 0) ^ ((y) >= 0)

/* Number of convolved data rates present in the table */
#define NUM_CONV_DATA (7)

/* List of data rates in bits/s whose corresponding pulse widths are */
/* convolved with the received data                                  */
int conv_data_rate[NUM_CONV_DATA] =
{1000, 50000, 100000, 150000, 200000, 250000, 300000};

/*****************************************************************************/
/* Function name   : hist_fn                                                 */
/* Input arguments : indata: pointer to input data containing pulse widths   */
/*                   indata_len: length of the buffer containg pulse widths  */
/*                   bin_data: pointer to the buffer containing the number   */
/*                      of occurances of pulse widths present in a histogram */
/*                      bin.                                                 */
/*                   mid_bin_data: data buffer containg the pulse width      */
/*                      corresponding to the middle of each bin              */
/* Output arguments: none                                                    */
/* Purpose         : This function determines the histogram of the pulse     */
/*                   widths present in indata using MAX_NUM_BINS number of   */
/*                   bins.                                                   */
/*****************************************************************************/
void hist_fn(int *indata, int indata_len, int *bin_data, float *mid_bin_data)
{
    float max_value, min_value; /* Largest and smallest data value */
    int   i; /* counter */
    float width; /* Width of each bin */
    float start_value; /* start value of each bin */

    /* Determine max and min values */
    min_value = indata[0];
    max_value = indata[0];
    for (i = 1;i < indata_len-1; i++)
    {
        if(indata[i] > max_value)
        {
            max_value = indata[i];
        }
        if(indata[i] < min_value)
        {
            min_value = indata[i];
        }
    }

    width = (max_value - min_value)/MAX_NUM_BINS;

    /* Initialize the bin_data to zero */
    for (i = 0; i < MAX_NUM_BINS; i++)
    {
        bin_data[i] = 0.0;
    }
```

```
    /* Determine the histogram bin entries */
    for (i = 0; i < indata_len; i++)
    {
        /* Index of bin to which the data belongs */
        int hist_idx = (int)floor((indata[i] - min_value)/width);

        if (hist_idx > MAX_NUM_BINS)
        {
            hist_idx = MAX_NUM_BINS;
        }
        /* bin_data[i] += 1; */
        bin_data[hist_idx] += 1;
    }

    start_value = min_value;

    /* Determine the mid value corresponding to each bin in the histogram */
    for (i = 0; i < MAX_NUM_BINS;i++)
    {
        mid_bin_data[i] = start_value + width/2;
        start_value += width;
    }
    return;
}
/*****************************************************************************/
/* Function name   : transit_detect                                          */
/* Input arguments : input: pointer to the buffer containing the filtered    */
/*                      data.                                                */
/*                   output: pointer to the buffer containing samples at     */
/*                      which transition has occured.                        */
/*                   output_length: number of elements in output buffer      */
/*                   input_size: number of elements in input buffer          */
/* Output arguments: none                                                    */
/* Purpose         :  Function that determines indices at which transitions  */
/*                    occur.  Since noise may cause spurious zero crossings, */
/*                    transitions are not recorded until they hold the same  */
/*                    sign for at least THRESHOLD samples.                   */
/*****************************************************************************/
void transit_detect(float *input, int *output, int *output_length,\
                    int input_size)
{
  int i, trans_count = 0;
  int state = STABLE, sign_count;

  for (i = 1; i < input_size; i++) {

    if (sign_change(input[i-1], input[i])){
      if (state == STABLE) {
        state = TENTATIVE;
        sign_count = 0;
      }
      else
        state = STABLE;
    }
    if (state == TENTATIVE && ++sign_count >= THRESHOLD) {
      state = STABLE;
```

```
      output[trans_count++] = i - THRESHOLD + 1;
    }
  }

*output_length = trans_count - 1;
}

/*****************************************************************************/
/* Function name   : filter_data                                             */
/* Input arguments : data : pointer to input data that is to be filtered.    */
/*                   data_len: length of the data buffer.                    */
/*                   filter_len: length of the filter.                       */
/*                   filter_hist_buf: pointer to the filter history buffer.  */
/*                   filtered_data: pointer to the buffer containing the     */
/*                      filtered data.                                       */
/* Output arguments: none                                                    */
/* Purpose         : This function filters the data using a moving average   */
/*                   filter of length filter_len                             */
/*****************************************************************************/
void filter_data(float *data, int data_len,
        int filter_len, float *filter_hist_buf, float *filtered_data)
{
    int i,j; /* counter */
    float sum;
    int index;

    /* Initialize the filter history buffer to zero */
    for (i=0; i < filter_len-1; i++)
    {
        filter_hist_buf[i] = 0;
    }

    for (i=0; i < filter_len; i++)
    {
        sum = data[i];

        for (j = 0; j < filter_len-1; j++)
        {
            sum += filter_hist_buf[j];
        }

        filtered_data[i] = sum;

        index = i%(filter_len-1);
        filter_hist_buf[index] = data[i];
    }

    for (i = i-1;i < data_len; i++)
    {
        sum = sum + data[i] - data[i - filter_len];
        filtered_data[i] = sum;
    }

    for (i = i-1;i < data_len + filter_len-1; i++)
    {
        sum = sum - data[i - filter_len];
        filtered_data[i] = sum;
```

```
    }
}

/*****************************************************************************/
/* Function name  : est_data_rate                                            */
/* Input arguments : sample_data_ptr: pointer to the sample_data structure   */
/*                   params_ptr: pointer to the acfgrx_params structure      */
/*                                                                           */
/* Output arguments: The function returns "0" for success and "-1" for       */
/*                      failure.                                             */
/* Purpose         : This is the function that does the estimation of data   */
/*                   rate.                                                   */
/*****************************************************************************/
int est_data_rate(sample_data_t *sample_data_ptr, acfgrx_params_t *params_ptr)
{
    int i; /* counter */
    int *pulse_widths; /* Array of pulse widths */
    float *filtered_data; /* transmitted data filtered using pulse */
    float *filter_hist_buf; /* Filter history buffer */
    int *index_buf; /* array containing indices at which transition occurs */

    /* The number of zeros observed in the histogram of a given data that is */
    /* filtered using different pulse widths                                 */
    int *num_zeros;

    /* This pointer points to array containing estimated data rates          */
    /* corresponding to different pulse widths                               */
    float *est_rate_list;

    /* Array containing number of occurances of data for each bin of the     */
    /* histogram                                                             */
    int *num_occurance;

    /* Array containing the middle value of each bin of the histogram */
    float *mid_bin_values;

    /* Considering mid_bin_data and num_occurance for histogram entries      */
    /* whose transition widths are greater than or equal to 50% of the       */
    /* convolved pulse width                                                 */
    float *valid_width;
    float *valid_occur;

    /* index corresponding to maximum number of zeros in the histogram */
    int max_zero_count = 0;
    int max_zero_index;

    /**********************/
    /* Allocate memory    */
    /**********************/
    num_zeros = (int*)malloc(sizeof(int) * NUM_CONV_DATA);

    if(num_zeros == NULL)
    {
        printf("Cannot allocate memory for num_zeros \n");
        return(-1);
    }
```

```
est_rate_list = (float*) malloc(sizeof(float) * NUM_CONV_DATA);
if(est_rate_list == NULL)
{
    printf("Cannot allocate memory for est_rate_list \n");

    /* De-allocate previously allocated memory */
    free(num_zeros);

    return(-1);
}

filtered_data = (float*)calloc(
    ceil(sample_data_ptr->rate/MIN_DATA_RATE) + sample_data_ptr->len - 1,
    sizeof(float));

if (filtered_data == NULL)
{
    printf("Cannot allocate memory to filtered_data \n");

    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    return(-1);
}

filter_hist_buf = (float*)malloc(sizeof(float) *
    (ceil(sample_data_ptr->rate/MIN_DATA_RATE)-1));

if (filter_hist_buf == NULL)
{
    printf("Cannot allocate memory to filter_hist_buf \n");

    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    free(filtered_data);
    return(-1);
}

index_buf = (int*) malloc(sizeof(int) * (sample_data_ptr->len - 1));
if(index_buf == NULL)
{
    printf("Cannot allocate memory to index_buf \n");

    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    free(filtered_data);
    free(filter_hist_buf);
    return(-1);
}

num_occurance = (int *) malloc(MAX_NUM_BINS * sizeof(float));
if (num_occurance == NULL)
{
    printf("Cannot allocate memory to num_occurance \n");
```

```
    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    free(filtered_data);
    free(filter_hist_buf);
    free(index_buf);
    return(-1);
}

mid_bin_values = (float *) malloc(MAX_NUM_BINS * sizeof(float));
if(mid_bin_values == NULL)
{
    printf("Cannot allocate memory to min_bin_data \n");

    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    free(filtered_data);
    free(filter_hist_buf);
    free(index_buf);
    free(num_occurance);
    return(-1);
}

valid_width = (float *) malloc(MAX_NUM_BINS * sizeof(float));
if(valid_width == NULL)
{
    printf("Cannot allocate memory to valid_width \n");

    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    free(filtered_data);
    free(filter_hist_buf);
    free(index_buf);
    free(num_occurance);
    free(mid_bin_values);
    return(-1);
}

valid_occur = (float *) malloc(MAX_NUM_BINS * sizeof(float));
if (valid_occur == NULL)
{
    printf("Cannot allocate memory to valid_occur \n");

    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    free(filtered_data);
    free(filter_hist_buf);
    free(index_buf);
    free(num_occurance);
    free(mid_bin_values);
    free(valid_width);
    return(-1);
}
```

```
/*************************************************************************/
/* Data rate estimation                                                  */
/*************************************************************************/
for (i = 0; i < NUM_CONV_DATA; i++)
{
    int k = 0; /* counter */
    int index_length;
    int valid_index = 0;
    int zero_count = 0;
    float max_valid_occur = 0;
    int max_index = 0;

    /* The data rate corresponding to the pulse width is considered */
    /* to be the maximum data rate                                  */
    int max_data_rate = conv_data_rate[i];

    /* length of the pulse */
    int filter_len = ceil(sample_data_ptr->rate/max_data_rate);

    filter_data(sample_data_ptr->data, sample_data_ptr->len,
        filter_len, filter_hist_buf, filtered_data);

    transit_detect(filtered_data, index_buf,
        &index_length, sample_data_ptr->len);

    pulse_widths = (int *)malloc(sizeof(int) * (index_length-1));

    if(pulse_widths == NULL)
    {
        printf("Cannot allocate memory to pulse_widths \n");

        /* De-allocate previously allocated memory */
        free(num_zeros);
        free(est_rate_list);
        free(filtered_data);
        free(filter_hist_buf);
        free(index_buf);
        free(num_occurance);
        free(mid_bin_values);
        free(valid_width);
        free(valid_occur);
        return(-1);
    }

    for (k = 0; k < index_length -1; k++)
    {
        pulse_widths[k] = index_buf[k+1] - index_buf[k];
    }

    /* Call the Histogram sub-routine */
    hist_fn(pulse_widths, index_length - 1, num_occurance,
        mid_bin_values);

    for (k = 0;k < MAX_NUM_BINS;k++)
    {
        if(mid_bin_values[k] >= (0.5 * filter_len))
        {
```

```
                valid_width[valid_index] = mid_bin_values[k];
                valid_occur[valid_index] = num_occurance[k];
                if(valid_occur[valid_index] > max_valid_occur)
                {
                    max_valid_occur = valid_occur[valid_index];
                    max_index = valid_index;
                }
                valid_index++;
            }
            if (num_occurance[k] == 0)
            {
                zero_count++;
            }
        }

        est_rate_list[i] = sample_data_ptr->rate/valid_width[max_index];
        num_zeros[i] = zero_count;
        if(num_zeros[i] > max_zero_count)
        {
            max_zero_count = num_zeros[i];
            max_zero_index = i;
        }
    }
    /* The data rate corresponding to maximum number of zeros is considered */
    /* to be the correct estimate of the data rate                          */
    params_ptr->data_rate = est_rate_list[max_zero_index];

    /* De-allocate previously allocated memory */
    free(num_zeros);
    free(est_rate_list);
    free(filtered_data);
    free(filter_hist_buf);
    free(index_buf);
    free(num_occurance);
    free(mid_bin_values);
    free(valid_width);
    free(valid_occur);
    free(pulse_widths);
    return (0);
}
/*****************************************************************************/
/* End of File                                                               */
/*****************************************************************************/
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-03-2005	Technical Report	01-05-2004 -- 01-05-2005

4. TITLE AND SUBTITLE

Annual Report: Research Supporting Satellite Communications Technology

5a. CONTRACT NUMBER

5b. GRANT NUMBER
NAG3-2864

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Horan, Stephen and Lyman, Raphael

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

New Mexico State University
Klipsch School of Electrical and Computer Engineering
Box 30001 / Dept. 3-0
Las Cruces, NM 88003-8001

8. PERFORMING ORGANIZATION REPORT NUMBER
NMSU-ECE-05-002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

NASA Goddard Space Flight Center
Code 567
Greenbelt, MD 20771
Technical Officer: David Israel

10. SPONSORING/MONITOR'S ACRONYM(S)
NASA/GSFC

11. SPONSORING/MONITORING REPORT NUMBER

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited; Distribution: Standard
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes the second year of research effort under the grant "Research Supporting Satellite Communications Technology," NAG5-13189. The research program consists of two major projects: Fault Tolerant Link Establishment and the design of an Auto-Configurable Receiver.
The Fault Tolerant Link Establishment protocol is being developed to assist the designers of satellite clusters to manage the inter-satellite communications. During this second year, the basic protocol design was validated with an extensive testing program. After this testing was completed, a channel error model was added to the protocol to permit the effects of channel errors to be measured. This error generation was used to test the effects of channel errors on Heartbeat and Token message passing. The C-language source code for the protocol modules was delivered to Goddard Space Flight Center for integration with the GSFC testbed.
The need for a receiver autoconfiguration capability arises when a satellite-to-ground transmission is interrupted due to an unexpected event, the satellite transponder may reset to an unknown state and begin transmitting in a new mode. During Year 2, we completed testing of these algorithms when noise-induced bit errors were introduced. We also developed and tested an algorithm for estimating the data rate, assuming an NRZ-formatted signal corrupted with additive white Gaussian noise, and we took initial steps in integrating both algorithms into the SDR test bed at GSFC.

15. SUBJECT TERMS

Space communications networks, Spacecraft communications, Digital communications systems, Signal processing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19b. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Stephen Horan
U	U	U	UU	91	19b. TELEPHONE NUMBER *(Include area code)* (505) 646-4856