Interplanetary Internet Routing System Documentation

Drafted by Sam Horovatin, co-authorship by Fredrik Johansson, Wai Chung Fung and Nicholas Brunoro

Contents

Project Abstract: 3

System Implementation: 4

- Terminology Definitions: 5
- System Description: 6
- Meta-Data Description: 9
- Project Scope: 9

Project Abstract:

At the current rate humanity is expanding into environments in which little to no networking infrastructure is present, and the general trend towards connectivity enjoyed by people and software systems today, the need for a communications system in these environments is high. Using a currently implemented system such as the deep space network designed for small scientific data transfers, the routing of internet traffic to a non-trivial number of devices would be impractical, highly latent, and lossy endeavor. As such, the goal of the project is to create a software system to facilitate this need for connectivity, and to design it in such a way that it is fault tolerant, capable of independent routing between uplink and downlink points, and to be easily expanded upon in future.

With this goal in sight, a first step would be to define the environment in which the system will operate in. First, it can be assumed the system is running on a finite resource hardware system with some digital transmission and receiving capabilities. For simplicity, aspects such as power consumption, hardware lifetime, transmission method, and transit times will be abstracted to their most basic forms. In a full-blown execution of the system these aspects would be critical to system success and would be specific to the hardware running it. These will not be addressed due to complexity and time constraints; however, consideration of these aspects will be considered in the systems design to allow future expansion and support.

The second major aspect of the environment would be its highly latent nature when it comes to communication over large distances. As the speed of light is only ~300 million m/s, and distances between planets such as earth and mars are in the hundreds of millions of kilometers, a transmission delay is expected. These latent delays would cripple protocols such as TCP and HTTP. Our system will provide a gate for these protocols to interact in these highly latent environments.

The last environmental aspect being directly addressed by our system is message loss and corruption. As a message is travelling a significant distance in an unpredictable environment, message loss or corruption is always a concern. Our system aims to provide considerations in its functionality to help alleviate the problems associated with message loss and corruption.

To further abstract and simplify the overall design, and to allow the system to achieve meaningful results in a simplistic manner, the environment will be transposed on to a 3-dimensional Euclidian space. The origin of this plane can be centered on anything significant in the environment, but for our system it will be centered upon the Sun. This environmental abstraction will be significant in the system's ability to route messages.

System Implementation:

With a well-defined system goal and environmental parameters, we can now move onto specifics about the systems implementation. First and foremost, the system will be implemented in Erlang. This is due to Erlang's ability to gracefully implement communication networks. The system's basic functions such as message handling, message routing, and system uptime/dependability lend themselves well to Erlang's syntax and design. Further-more with Erlang's built in ability to handle specific hardware and software failures effectively, efficiently run concurrent operations, and its general speed and reliability, we deemed it a good fit for this project.

Terminology Definitions:

Prior to defining any system specifications some basic terminology should be laid out to allow maximum clarity:

Node: In the context of this project, a node is a single instance of the system within the network. This concept can be specified to a single computer running a single instance of the system and thus acting as a single node, or a single computer running multiple instances of the system and thus hosting multiple nodes. This definition stems from graph traversal used to route messages in the network.

Uplink: The act of transmitting a message from a terrestrial or otherwise stable network to the system of satellites (less stable network): Can be considered a start node in the network.

Downlink: The act of transmitting a message from the system of satellites (less stable network) to a terrestrial or otherwise stable network: Can be considered an end node in the network.

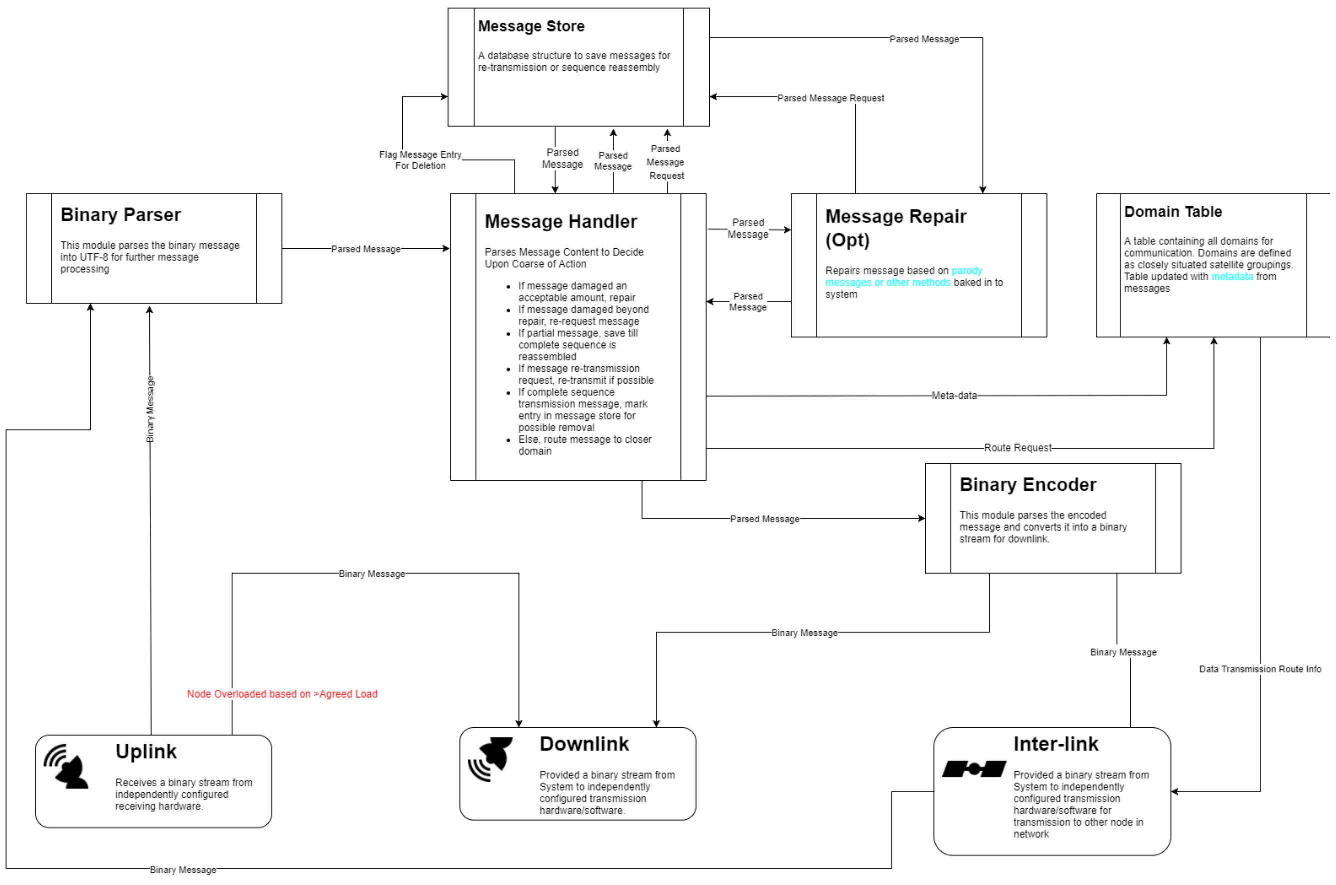
Inter-link: The act of transmitting a message from one node in system of satellites to another node in the system of satellites.

Domain: A grouping of nodes that are separated by relativistically trivial distances comparative to other nodes in the known network. This grouping is based on nodes relevant distance alone, and thus nodes can be members of multiple domains at a single time.

Message: Loosely used to describe any communication between nodes. This includes the meta-data tag and actual data being sent between nodes.

System Description:

What follows is an abstract description of all portions of the system, separated into module groupings. It should be noted that in Figure 1, the Uplink, Downlink, and Inter-link are not modules but separate software implementations feeding in or consuming a binary data stream produced from either hardware drivers or the systems binary encoder. These portions of the system would be specific to hardware configurations and thus are not included in our implementation. For our project, a binary output/input can be assumed based on the current telecommunication data encoding standards.



Message Store
A database structure to save messages for re-transmission or sequence reassembly
Parsed Message
Parsed Message Request
Flag Message Entry For Deletion
Parsed Message
Parsed Message
Parsed Message Request
Binary Parser
This module parses the binary message into UTF-8 for further message processing
Parsed Message
Message Handler
Parses Message Content to Decide Upon Coarse of Action
If message damaged an acceptable amount, repair
If message damaged beyond repair, re-request message
If partial message, save till complete sequence is reassembled
If message re-transmission request, re-transmit if possible
If complete sequence transmission message, mark entry in message store for possible removal
Else, route message to closer domain
Parsed Message
Parsed Message
Message Repair (Opt)
Repairs message based on parody messages or other methods baked in to system
Domain Table
A table containing all domains for communication. Domains are defined as closely situated satellite groupings. Table updated with metadata from messages
Meta-data
Route Request
Binary Message
Binary Encoder
This module parses the encoded message and converts it into a binary stream for downlink.
Parsed Message
Binary Message
Binary Message
Binary Message
Data Transmission Route Info
Node Overloaded based on >Agreed Load
Uplink
Receives a binary stream from independently configured receiving hardware.
Downlink
Provided a binary stream from System to independently configured transmission hardware/software.
Inter-link
Provided a binary stream from System to independently configured transmission hardware/software for transmission to other node in network
Binary Message

Figure 1: General overview of project module connection

Binary Parser:

Dependency: A Binary Data Stream

Return: A message that has its metadata encoded in UTF-8 with message data remaining un-encoded.

Description: The general raw bit handler for the system. Takes the raw binary stream produced by the hardware translation of telecommunication encodings and encoded it into an encoding schema that the system can is able to parse further within the system (Erlang primarily works with the UTF-8 encoding pattern).

Binary Encoder:

Dependency: A message that has its metadata encoded in UTF-8 with message data remaining un-encoded

Return: A Binary Data Stream

Description: The system's binary translator. It takes a message which is encoded in UTF-8 and converts it to a binary stream which the transmission hardware and drivers can work with.

Message Handler:

Dependency: A parsed and encoded message, access to Message Storage, and message repair functions.

Return: Correct course of action for message routing based on message meta-data

Description: The main functioning unit of system. Takes message metadata and determines ultimate course of action for message. It will also provide meta-data for domain table updates. The actions the Message handler can take are:

- Request a message resend
- Store message for further transmission
- Mark stored message for deletion based on successful transmission
- Request message from message store for re-transmission
- Repair message based on damage
- Request a route for message and send message to Binary encoder
- Request a route for message and send successful message transmission message to Binary encoder

Message Store:

Dependency: A parsed and encoded message, knowledge on drive sizes and usage

Return: Storage entry

Description: This module is the long term, main non-volatile storage of messages in the system. Resources are managed based on time, with oldest or flagged messages being deleted first upon storage necessity. It primary function is to store entire messages for future transmission. Once a message has been successfully transmitted to another node, all entries in the table are flagged for potential deletion.

Message Repair:

Dependency: A message that has its metadata encoded in UTF-8, with message data remaining un-encoded.

Return: A message that has its metadata encoded in UTF-8, with message data remaining un-encoded.

Description: A module containing a set of message repair algorithms to aid in message accuracy. This module will not be fully implemented in project due to complexity and time restraints.

Domain Table:

Dependency: Metadata from every message received, Route Request

Return: A next node to route message to

Description: A module with a hybrid non-volatile memory and volatile memory data table of all known satellites. When requested, it will provide a best next route based on its internal table state. This next step is found using a simplified path finding algorithm, as a node will always communicate directly with the destination if destination node is visible. In cases where the destination is unknown, it will make pass the message to the closest node.

Meta-Data Description:

Below is the breakdown of the meta-data info encapsulating every message routed into the system:

The metadata structure is contained at the head of every message while data is contained at the end. Due to the amount of dynamic data, a fixed length header for the metadata is not possible. Therefore, for simplicity and flexibility, the metadata uses tags to define the information enclosed in it. The metadata is encoded in UTF-8 and uses the same system as markup languages. This allows information to be nested if necessary and new or optional tags can be introduced in the future.

An example metadata format:

```
<metadata>
        <sender>senderID</sender>
        <dest>destinationID</dest>
        <nodesTouched>Satellite1ID, Satellite2ID, Satellite3ID</nodesTouched>
        <msgID>101</msgID>
        <packetID>4</packetID>
        <msgSize>50</msgSize>
        <dataEncoding>UTF-8</dataEncoding>
        <errorCorrectInfo></errorCorrectInfo>
</metadata>
<data>
Hello World
</data>
```

Project Scope:

The project is scoped as follows, with all other actions being out of scope or not part of the project:

- Creation of all previously listed modules except for the message repair module, which will be partially implemented due to complexity.
- The ability to pass messages in-between at least 5 nodes, without message loss
- An overall system ability to handle failure gracefully, gauged by persistent function when hardware and software failures are occur or are simulated
- Utilization of Erlang's live code-hot swap to update the code while still functioning.

What is not in scope for the project is as follows, with no intention or resources to be applied to them:

- More than 80% uptime capabilities
- Message repair algorithms
- Variable load balancing
- Power critical, transmission medium specific, future line of site, or any other specialized routing