Analyzing Network Transfer Protocols on a Simulated Interplanetary Internet

Cody Auch, Mark Lysack, Jacob Janzen

April 8, 2024

Abstract

TODO insert abstract here

1 Introduction

In the past decade interest in returning back to space has increased internationally. Many organizations have started sending remote vehicles into space for the first time and there are talks of a permanent lunar base being created. With an increased amount of satellites and potentially permanent bases outside of Earths orbit, the demand on the interplanetary network (IPN) is increased. The IPN has a series of unique challenges such as a high amount of error, constantly changing network topology, and long propagation delay. The solution to these problems is implementing a delay tolerant network. While space organizations have maintained the IPN with a set of standard protocols, these may not be sufficient for an expanding network. The simulator proposed shows how an IPN can handle different stages of development of an IPN as well as increased throughput requirements.

2 Previous Work

3 Methodology

3.1 Physics Engine

A significant challenge in creating a IPN is the rapidly changing topology. To simulate these rapid changing conditions a physics engine was created to model the orbits of several entities in the solar system. Several assumptions

were made in the simulator for the sake of simplicity as the goal of the simulator is not to create a perfect model of the solar system and the entities inside of the solar system, but rather conditions which a IPN would exist in.

The first assumption made was that all orbits are circular. Realistically, orbits are oval shaped, with the two focal points. A result of this assumption is the elimination of any difference in orbital speed due to Kepler's third law. The orbital radius was chosen by taking the average orbital radius of the real life celestial bodies. For example Earth's average orbital radius is 149.6 billion meters [**<empty citation>**], and therefore orbits around the sun in a circular orbit of radius 149.6 billion meters. Another result of this assumption is that orbits are considered to be centered around the body which it orbits. The Mars orbit has a focal point that is further in space, making it more oval shaped than the earth orbit. [**<empty citation>**] While this could make changes to a simulated network in some very specific situations, the primary behaviour of a solar system is maintained, like retrograde orbits, periodical alignment, predictable behaviour, and recursive orbits of sub orbiting object.

All entities are assumed to have a circular shape, with a constant radius. Due to the scale of the objects, both large and small, this has a very minor effect.

The whole physics simulation has been done on a 2D plane. While planetary orbits were not perfectly aligned, on the astronomical scale which we are working on, assuming that all their orbits are contained on the same plane will not effect the simulation significantly. Where this could be an issue is for when there are more than two satellites orbiting the same entity, with a relatively low orbit. For example satellites orbiting Earth. While there are some complex satellite interactions which are removed it does still simulate satellites having to form a network around the entity it is orbiting, it just requires less satellites then if it needed to form a 3-D network.

Messages between entities were assumed to be sent at the speed of light, as all forms of radio, lasers, or other wireless messages all are sent at the speed of light. Since propagation delay is much higher then terrestrial networks, other delays (TODO specify which delays) were ignored as being negligible.

Based on these simplifications, a physical simulation with several goals. To calculate if a line of sight could be made between two communicating entities, that is if a message can be sent, to calculate the error rate when sending a message, and to calculate the message propagation delay for the message to be sent. Line of sight was achieved using simple geometry calculations. (See appendix) All entities are able to block signals if the ray passes through the radius of the entity. One of the largest factors contributing to transmission error is the distance travelled by the electromagnetic signal between transmitting and receiving nodes. By generalizing Friis' transmission

formula for isotropic emitters and receivers, we can obtain the ratio of signal intensity received to signal intensity transmitted, known as Free Space Path Loss (FSPL). We can use FSPL as our transmission error rate, giving the following expression: $ErrorRate = (\frac{4\pi d}{\lambda})^2$ where d is the distance between nodes in metres and λ is the wavelength of the electromagnetic wave used in transmission. As much of space communications utilizes the Ka-band of frequencies (27–40 GHz) we opted to use a transmission frequency of 30Ghz. The transmission wavelength can be then calculated according to $\lambda = \frac{c}{f}$, where c is the speed of light and f is the transmission frequency, giving a transmission wavelength of about 9.99 mm. Propagation delay is a simple function of distance and the speed of light. The input of the physics engine is a time t from some preset initial conditions which outputs the state of all entities.

3.2 Network Simulation

We used NS-3 [1] to implement a simulation of this network. Our final design consisted of a set of routers representing the different entities in our physics simulation which were each linked to every other router using a point-to-point channel. Essentially, this allowed us to have a wired network with configurable delay and error along with the ability to take certain channels down if they are obstructed that we could make behave like an interplanetary network would without having to deal with the complexities of configuring wireless communication. In addition to the mesh of routers, we added two individual nodes to the network with zero delay that can be connected to any one router in the network. One of these two nodes acts as a sender and the other one acts as a receiver. This configuration of sender and receiver as separate nodes made it easier to decouple the applications from the network topology and allowed us to programatically create arbitrary topologies.

NS-3 is a useful library, but we did run into a number of problems over the course of our implementation of our simulation. The first problem came when we first tried to compile the library itself and ran into a bug in the build script that caused all executables to fail to run if there was the word "scratch" anywhere in the path due to a specific subdirectory of NS-3 which also happens to be called "scratch". When first testing it out, one of us had put NS-3 into a directory that happened to contain that word. We made a pull request to fix this issue and it was merged by the NS-3 maintainers which allowed us to continue.

The next issue we ran into was that we had decided to use the Python bindings to allow us to easily use Python for the physics simulation. This was a mistake. Although NS-3 does have Python bindings that do make it

easier to run it as a script after writing, there is no documentation at all for the Python bindings and some basic features such as setting the error rate on a channel or scheduling a task to run during the simulation have no Python support and must use inline C++. The lack of support for task scheduling in Python is particularly bad as it forced us to schedule a C++ function which itself called Python code. It also forced us to use global state as `CPyCpyy` does not have support for arbitrary function parameters when calling Python code as far as we could tell (this feature is not documented and we only figured out about it because the few Python examples in the NS-3 source code that do use scheduling also call Python from C++ with this method).

Before we settled on using point-to-point to emulate a wireless network, we attempted to use the existing wireless networking features of NS-3. These features are very powerful and even have a positioning and velocity system for nodes in the network which would be very useful. We ran into a lot of issues with this system though. Firstly, there only seems to be real support for Wi-Fi which is not particularly useful for long distance unlike radio. This meant that we would have to greatly decrease the distances that we were simulating. This is fine and would still allow a perfectly reasonable simulation so long as we increased the distances again in our computations while analyzing the data. The other problem that we ran into is the fact that the wireless networking would quickly require us to go well outside the scope of our project. Determining the gain and energy required to configure our satellites to send and receive messages correctly is more of an engineering problem that we were not interested in solving.

After settling on using point-to-point, UDP was easy to get working but TCP proved to be very challenging. Our initial implementation of our topology seemed to have an issue where messages could be routed in one direction but not the other. This meant that when TCP tried to establish a connection it would always fail. It still is not entirely clear what was configured incorrectly, but after changing how we were creating channels, TCP was finally working. We ran into more issues with TCP though, when we started adding delay to the channels. It turns out that NS-3 sockets have a rather low time-to-live and there do not seem to be any global options for configuring sockets. We were left with two options: implement TCP from scratch using custom-built sockets for each node in the network or ensure that the messages that we do send have lower delay than the TTL for our sockets. We opted for the latter. To make sure that TCP messages could successfully ACK, we found a value that we could divide all times by to make sure that almost every message that could feasibly be sent would be received before its TTL expired. This does not change any distances or error rates and effectively only changes the unit we use for time from s to $\frac{s}{26}$ which has an easy conversion back to s. Finally,

we implemented a version of our program using the New Reno variant of TCP. Thankfully, NS-3 made this easy for us. It worked perfectly after assigning a configuration value before setting up our communication channels.

References

[1] nsnam. *NS-3*. Version 3.41. Feb. 10, 2024. URL: `https://www.nsnam.org/`.