UTN FACULTAD REGIONAL MENDOZA
UNIVERSIDAD TECNOLÓGICA NACIONAL

MAESTRÍA EN INSTRUMENTOS SATELITALES

TESIS

"Verificación y Análisis Funcional de Algoritmos de Enrutamiento para Instrumentos Satelitales Distribuidos"

Ing. Vega Molinas, Blas Fernando

UFS Unidad de Formación Superior

CONAE

Córdoba, 20 de abril de 2023

TESIS DE MAESTRÍA EN INSTRUMENTOS SATELITALES

Verificación y Análisis Funcional de Algoritmos de Enrutamiento para Instrumentos Satelitales Distribuidos

Ing. VEGA MOLINAS, Blas Fernando
Tesista

Dr. Fraire, Juan Andrés
Director

Miembros del Tribunal revisor
Dr. Pablo Madoery
M.Sc. Pablo Soligo
Dr. Pablo Servidia

20 de Abril de 2023

Redes Tolerantes a Retrasos en pequeños satélites.

Unidad de Formación Superior
Universidad Tecnológica Nacional – Facultad Regional Mendoza
Comisión Nacional de Actividades Espaciales
Argentina

En especial agradecimiento a la Comisión Nacional de Actividades Espaciales, por darme la oportunidad de crecer profesionalmente, entablar lazos de amistad y expandir mis límites. También a los hermanos argentinos, pueblo del cual me siento orgulloso por toda la dedicación y esmero que ponen en cada paso que dan.

Tabla de Contenidos

Tabla de Contenidos VII

Lista de Tablas IX

Lista de Figuras XI

Lista de Acrónimos XIII

Resumen XVII

Abstract XIX

Agradecimientos XXI

1. Introducción 1

1.1. Justificación del Estudio Realizado . . . 2
1.2. Hipótesis . . . 4
1.3. Interrogantes de la Investigación . . . 4
1.4. Objetivos . . . 5
1.4.1. Objetivos Generales . . . 5
1.4.2. Objetivos Específicos . . . 5

2. Estado del Arte 7

2.1. Comunicaciones Satelitales . . . 8
2.1.1. Historia . . . 8
2.2. Desafíos en las Comunicaciones Satelitales . . . 11
2.2.1. Utilización de COTS en Satélites . . . 12
2.3. Redes Terrestres TCP/IP . . . 14
2.4. Redes Satelitales . . . 15
2.4.1. Redes Tolerantes a Retrasos (DTN) . . . 16
2.5. Aspectos Generales . . . 17
2.5.1. Paquete (Bundle) . . . 17
2.5.2. Bundle Protocol (BP) . . . 18
2.6. Enrutamiento en Redes Tolerantes a Retraso . . . 20
2.7. Red de Superposición Interplanetaria, (ION) . . . 24
2.8. Enrutamiento Mediante Grafos de Contacto (Contact Graph Routing) . . . 26
2.8.1. Historia . . . 26
2.8.2. Contactos . . . 27
2.8.3. Estructura de un Contacto . . . 28
2.8.4. Plan de Contactos . . . 28
2.8.5. Grafos . . . 30
2.8.6. Grafo de Contactos . . . 30
2.8.7. Enrutamiento (Procedimiento) . . . 31

2.9. Algoritmo CGR 33
2.9.1. Algoritmo de Dijkstra 33
2.9.2. Algoritmo de Procedimiento de Revisión de Contactos CRP 34
2.9.3. Algoritmo de Procedimiento de Selección de Contactos CSP 34

3. Implementación **37**
3.1. Características de la NanoMind A712c 37
3.1.1. Procesador 39
3.1.2. Memoria 40
3.1.3. Conectores 42
3.1.4. Protocolos de Comunicación 42
3.1.5. Herencia de Vuelo 44
3.1.6. Sistema Operativo 44
3.1.7. Manejo de Memoria en FreeRTOS 49
3.2. Implementación del CGR en la NanoMind 49
3.2.1. Creación de la Tarea CGR 50
3.2.2. Estructura 51
3.2.3. Estructura Principal 52
3.2.4. Estructura de *cgrForward* 58
3.2.5. Estructura de Identificación de Nodos 60
3.2.6. Estructura de Carga de Rutas de Paquete *loadRouteList* 62
3.2.7. Estructura de Selección *firstDepleted* 62
3.2.8. Estructura Algoritmo de Dijkstra 64
3.2.9. Estructura de Rutinas de Manejo de Listas Enlazadas 68
3.3. Carga del algoritmo en la NanoMind 70
3.4. Campaña Experimental 71

4. Resultados **73**
4.1. Resultados - Subrutina CGR 73
4.2. Análisis de Resultados 74
4.2.1. Tiempo de Ejecución 74
4.2.2. Análisis de Memoria 76
4.2.3. Ajuste de Curvas de los Resultados 81

Conclusiones **85**

Trabajos futuros y recomendaciones **87**

Anexos **89**
A. Plan de Contactos Utilizado 91

Referencias bibliográficas **95**

Lista de Tablas

2.1. Elementos de datos del paquete (Relevante para el enrutamiento) 17
2.2. Parámetros de un Contacto 29
2.3. Ejemplo de un Plan de Contactos 30
2.4. Parámetros de Rutas 33

4.1. Cantidad de rutas encontradas por cantidad de contactos agregados, para los nodos 14, 19, 28 y 13 74
4.2. Cantidad de Llamadas a Cada Función 76
4.3. Mediana de Tiempos de Ejecución (en segundos) encontradas por cantidad de contactos agregados al plan] 81

Lista de Figuras

2.1. Ejemplo de almacenamiento, transporte y reenvío en una DTN espacial. Un nodo en la Tierra transmite datos a un satélite en órbita alrededor de la Luna, que los almacena hasta que esté disponible un contacto de largo alcance con Marte. Debido a a) los retrasos y b) las interrupciones, la dependencia de los mensajes de retroalimentación inmediata debe reducirse o eliminarse de las DTN. Dado que un segundo-luz se define como la distancia que recorre la radiación electromagnética en el espacio libre en un segundo, es una unidad de distancia que se corresponde trivialmente con un segundo-luz. distancia que corresponde trivialmente a una unidad de tiempo. 18

3.1. Entorno de desarrollo para C/C++, Eclipse 38
3.2. NanoMind A712c 38
3.3. Diagrama de bloque de la NanoMind A712c 40
3.4. Conectores en la placa electrónica NanoMind A712c 41
3.5. Setup de la campaña experimental. Laboratorio de la Unidad de Formación Superior UFS-CONAE. 71
3.6. Plan de contactos utilizado para las pruebas. 72

4.1. Tiempo de ejecución de CGR para distintas cantidades de contactos agregados al plan. 75
4.2. Memoria utilizada durante la ejecución de CGR, para distintas cantidades de contactos agregados al plan 77
4.3. Eficiencia de uso de memoria 81
4.4. Curva que representa el desempeño del algoritmo CGR corriendo en la OBC NanoMind. 83
4.5. Curva que representa la Memoria Utilizada por el algoritmo CGR corriendo en la OBC NanoMind. 84

Lista de Acrónimos

ADCS	Subsistema de Control y Determinación de Actitud (*Atittude Determination and Control Subsystem*).
AMS	Servicio de Mensajes Asíncronos (*Asynchronous Message Service*).
AP	Punto de acceso (*Access Point*).
API	Interface de Programación de Aplicación (*Application Programming Interface*).
ARM	Máquina RISC Avanzada (*Advanced RISC Machine*).
ASI	Agencia Espacial Italiana (*Agenzia Spaziale Italiana*).
ASIC	Circuito integrado para aplicaciones específicas (*Application-specific Integrated Circuit*).
BGA	Arreglo Matricial de Bolas (*Ball Grid Array*).
BP	Protocolo de paquetes (*Bundle Protocol*).
BPA	Agente de protocolo de paquetes (*Bundle Protocol Agent*).
BSR	Registro de Escaneo Límite (*Boundary Scan Register*).
BSS	Servicio de Transmisión del Paquete (*Bundle Streaming Service*).
CCSDS	Comité Consultivo de Sistemas de Datos Espaciales (*Consultative Committee for Space Data Systems*).
CFDP	Protocolo de Entrega de Archivo CCSDS (*CCSDS File Delivery Protocol*).
CGR	Enrutamiento mediante grafos de contacto (*Contact Graph Routing*).
CLA	Adaptadores de capas de convergencia (*Convergence Layers Adapters*).
COM	Subsistema de Comunicaciones (*Communication Subsystem*).
CONAE	Comisión Nacional de Actividades Espaciales.
COTS	Componentes estándares del mercado (*Components-of-the-Shelf*).
CPU	Unidad Central de Procesamiento (*Central Processing Unit*).
CRP	Procedimiento de revisión de contactos (*Contact Review Procedure*).
CSP	Procedimiento de selección de contactos (*Contact Selection Procedure*).
DGR	Retransmisión de Datagrama (*Datagram Retransmision*).

DSP Procesador de Señales Digitales (*Digital Signal Processor*).

DTN Red tolerante al retraso (*Delay Tolerant Network*).

DTPC Acondicionamiento de Carga Útil Tolerante a Retraso (*Delay Tolerant Payload Conditioning*).

ECC Capacidad estimada de consumo (*Estimated Capacity Consumption*).

EPS Subsistema de Potencia Eléctrica (*Electrical Power System*).

ESA Agencia Espacial Europea (*European Space Agency*).

ESPACENET Marco de Redes de Sensores Evolucionables y Reconfigurables para el Monitoreo y el Diagnóstico Aeroespaciales (*Envolvable Networks of Intelligent and Secure Integrated and Distributed Reconfigurable System-On-a-Chip*).

FBP Protocolo de reenvío de paquetes (*Forward Bundle Protocol*).

FPGA Matriz de Puertas Lógicas Programable en Campo (*Field-Programmable Gate Array*).

FreeRTOS Sistema Operativo en Tiempo Real Libre (*Free Real Time Operating System*).

FTDI Dispositivos Tecnológicos Futuros Internacionales (*Future Technology Devices International Ltd*).

FTP Protocolo de Transferencia de Archivos (*File Transfere Protocol*).

GEO Órbita Ecuatorial Geosincrónica o Geoestacionaria (*Geosynchronous -or Geostationary- Equatorial Orbit*).

GMES Monitoreo global del Medio Ambiente y la Seguridad (*Global Monitoring for Environment and Security*).

GPS Sistema de Posicionamiento Global (*Global Position System*).

I2C Circuito Inter-Integrado (*Inter-Integrated Circuit*).

ICI Infraestructura de Comunicación Interplanetaria (*Interplanetary Communication Infrastructure*).

IEEE Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers*).

IETF Grupo de Trabajo de Ingeniería de Internet (*Internet Engineering Task Force*).

ION Red de Superposición Interplanetaria (*Interplanetary Overlay Network*).

IP Protocolo de Internet (*Internet Protocol*).

IPN Red de Superposición Interplanetaria (*Interplanetary Network*).

IRTF Fuerza de Tarea de Investigación en Internet (*Internet Research Task Force*).

ISL Enlace entre Conmutadores (*Inter-Switch Link*).

JPL Laboratorio de Propulsión a Reacción (*Jet Propulsion Laboratory*).

JTAG Grupo de Acción de Ensayo Conjunto (*Joint Test Action Group*).

LEO Órbita terrestre baja (*Low Earth Orbit*).

LTP Protocolo de Transmisión Licklider (*Licklider Transmission Protocol*).

MIT Instituto Tecnológico de Massachusetts *Massachusetts Institute of Technology*.

MOC Centro de Operación de Misión (*Mision Operation Center*).

MTU Maximun Transmission Unit (*Unidad de Transmisión Máxima*).

OBC Computadora de Abordo (*On-Board Computer*).

OSI Modelo de Interconexión de Sistemas Abiertos (*Open Systems Interconnection*).

OWLT Tiempo Unidireccional de Luz(*One Way Light Time*).

PER Tasa de error de paquete (*Packet Error Rate*).

RAM Memoria de Acceso Aleatorio (*Random Access Memory*).

RH Endurecimiento a Radiación (*Radiation Hardening*).

RHBD Endurecimiento a Radiación por Diseño (*Radiation Hardening by Design*).

RISC Computadora con Conjunto Reducido de Instrucciones (*Reduced Instruction Set Computer*).

RS232 Estandar Recomendado 232 (*Recommended Standard 232*).

RTOS Sistema Operativo en Tiempo Real (*Real Time Operating System*).

RTT Tiempo de Ida y Vuelta (*Round Trip Time*).

SABR Enrutamiento de Paquetes Según Programa (*Schedule-Aware Bundle Routing*).

SAR Radar de Apertura Sintética (*Synthetic Aperture Radar*).

SARE Satélite de Alta Revisita.

SD Dispositivo de Almacenamiento (*Storage Device*).

SEFI Interrupción funcional de evento único (*Single Event Functional Interrupt*).

SEL Corto de un solo evento (*Single Event Effects*).

SEU Seteo de un solo evento (*Single Event Effects*).

SIASGE Sistema Italo Argentino de Satélites para la Gestión de Emergencias.

SPI Interface Periférica Serial (*Serial Peripheral Interface*).

TCP Protocolo de Control de Transmisión (*Transmission Control Protocol*).

TID Dosis Ionizante Total (*Total Ionizing Dose*).

TTL Tiempo de Vida (*Time To Live*).

UART Transmisor-Receptor Asíncrono Universal (*Universal Asynchronous Receiver-Transmitter*).

UDP Protocolo de Datagrama de Usuario (*User Datagram Protocol*).

UFS Unidad de Formación Superior.

UHF Ultra Alta Frecuencia (*Ultra High Frequency*).

USART Transmisor-Receptor Síncrono-Asíncrono Universal (*Universal Synchronous-Asynchronous Receiver-Transmitter*).

WCET Tiempo de Ejecución del Peor Caso (*Worst Case Execution Time*).

WLAN Red de Área Local (*Wireless Local Área Network*).

Resumen

El sector espacial se enfrenta a una evolución sin precedentes hacia plataformas satelitales distribuidas que, a pesar de estar compuestas por vehículos espaciales con recursos limitados, pueden ofrecer resultados de misión similares a los esperados de plataformas de gran tamaño. En el caso de la CONAE -la Agencia Espacial Argentina- este concepto se plasma en la "arquitectura segmentada", que está en sintonía con el programa de lanzadores Tronador que se está desarrollando en ese país. Entre los muchos retos que impone una arquitectura distribuida de satélites, esta tesis se centra en los mecanismos de red encargados de decidir las rutas de reenvío óptimas para encaminar los datos en un sistema compuesto por enlaces espacio-tierra y espacio-espacio. En efecto, el reducido número, tamaño, peso y potencia de las naves espaciales constituyentes, pequeños y nano-satélites, combinado con la dinámica orbital rápidamente cambiante, hace que las oportunidades de transferir datos (contactos) estén limitadas en el tiempo. Como resultado, los satélites necesitan calcular rutas basadas en contactos que tengan en cuenta el almacenamiento temporal de datos en nodos intermedios: un paradigma de red conocido como Red Tolerante al Retraso (DTN) [1]. El principal aporte de esta tesis es la implementación y evaluación del mecanismo de enrutamiento DTN más avanzado, conocido como *Contact Graph Routing* (CGR) [2]. Desarrollado y validado en vuelo por el JPL (NASA), CGR es el esquema de enrutamiento punta de lanza para futuras arquitecturas distribuidas con enlaces retrasados/interrumpidos. Sin embargo, ha sido criticado por su complejidad y limitada escalabilidad. Este trabajo es el primero en implementar y evaluar el rendimiento de CGR en un ordenador de vuelo *NanoMind* A712c diseñado para nano-satélites disponible en los laboratorios del Instituto Gulich en CONAE. Los resultados proporcionan una valiosa evidencia de las limitaciones de las topologías de flota que pueden ser consideradas para futuras arquitecturas segmentadas basadas en nano-satélites. También motivan futuras investigaciones para mejorar la escalabilidad y eficiencia computacional de CGR, un sector activo en la comunidad de redes espaciales. Los resultados obtenidos en esta tesis se han publicado en la conferencia *SmallSat* 2020 [3] con escrutinio y comentarios positivos de destacados revisores del sector. xvii

Abstract

The space sector is facing an unprecedented move toward distributed satellite platforms that, despite being comprised of resource-constrained spacecraft, can deliver mission outcomes similar to those expected from large-sized platforms. In the case of CONAE -the Argentinian Space Agency- this concept is mapped into the "segmented architecture", which is in sync with the Tronador launcher program under development in that country. Among the many challenges a distributed satellite architecture imposes, this thesis focuses on the networking mechanisms in charge of deciding optimal forwarding paths to route data in a system comprised of space-to-ground and space-to-space links. Indeed, the reduced number, size, weight, and power of constituent small and nano-satellite spacecraft, combined with the fastly changing orbital dynamics renders time-bounded opportunities for transferring data (contacts). As a result, satellites need to compute routes based on contacts that consider temporal data storage in intermediate nodes: a networking paradigm known as Delay-Tolerant Networking (DTN)[1]. The core contribution of this thesis is the implementation and evaluation of the state-of-the-art DTN routing mechanism known as Contact Graph Routing (CGR)[2]. Developed and flight-validated by JPL (NASA), CGR is the spearhead routing scheme for future distributed architectures with delayed/disruptive links. However, the approach has been criticized for its complexity and limited scalability. This work is the first to implement and evaluate the performance of CGR in a NanoMind A712c flight computer designed for nano-satellites available in the laboratories of the Gullich institute in CONAE. Results [3] provide valuable evidence of the limitations of the fleet topologies that can be considered for future nano-satellite-based segmented architectures. They also motivate future research on improving CGR scalability and compute efficiency, an active sector in the space networking community. The results obtained in this thesis have been published in the SmallSat 2020 conference [3] with scrutiny and positive feedback from leading reviewers from the industry.

Agradecimientos

Texto de los agradecimientos

A CONAE
A mis compañeros de la MIS
A los profesores de la MIS
A toda la familia de la UFS/Gulich
Al Dr. Juan Fraire
Al Dr. Jorge Kurita
A mi querida
A mi familia

CAPÍTULO 1

Introducción

Debido al aumento de la demanda de servicios de alta complejidad y el desarrollo tecnológico, han disminuido los costos de producción, el consumo de energía y se ha aumentado la frecuencia de ciclos de reloj de los circuitos integrados, la cantidad de núcleos de procesamiento como también la capacidad de almacenamiento mediante memorias *FLASH*[4]. Como consecuencia de la miniaturización e integración, ha aumentado la tasa de datos generados por instrumentos utilizados en misiones satelitales [5].

La velocidad de estos avances da lugar al aumento en el riesgo de fallas. Para garantizar la confiabilidad en el desempeño de estos componentes, es necesario invertir en el estudio de sus características de funcionamiento y conocer aspectos críticos que ayuden a tomar decisiones en el diseño de la misión.

Estas tendencias tecnológicas han permitido nuevas oportunidades en el mercado espacial y la puesta en marcha de proyectos aplazados o suprimidos durante años debido a los elevados costes inherentes. Y lo que es más importante, estos logros tecnológicos han originado una nueva fiebre espacial, con el lanzamiento de cientos de pequeños satélites en los últimos años y la puesta en servicio de un número aún mayor de ellos en un futuro próximo.

Las redes satelitales se volvieron la tendencia en cuanto a comunicaciones se refiere. Se han puesto en marcha proyectos con miles de satélites en órbitas bajas que permiten el envío de datos a baja latencia, alta velocidad y confiabilidad en poblaciones aisladas, comunidades rurales, o zonas en conflicto político o simplemente zonas donde las redes existentes no son económicas o son poco confiables.

Se han lanzado progresivamente constelaciones de satélites pensadas para Internet. Estas constelaciones están densamente pobladas ("constelaciones densas"), para garantizar un enlace constante entre los nodos de una red. Se puede citar como ejemplo a *SpaceX*, *Amazon*, *OneWeb*, que invierten en tecnologías de telecomunicación con propósitos similares, con el objetivo de aumentar el flujo de datos en los dispositivos que utilizamos día a día, mientras aumenta la diversidad de dispositivos que necesitan esta conexión constante a la red [6].

Por otra parte, existen propuestas de redes para escenarios donde la conectividad no es estable, pero toleran la conexión intermitente de los nodos. Estas redes son llamadas Redes Tolerantes a Retraso (DTN, del inglés *Delay Tolerant Networks*) [1], que a diferencia de las redes densas, están pensadas para constelaciones "esparsas" con mucha menor cantidad de satélites, donde la misión es el envío y recepción de datos. Los contactos pueden determinarse mediante cálculos de propagación de órbita, como también pueden ser oportunisticos en nodos muy específicos, en estos casos necesitan enrutamientos que respeten ese paradigma. Si los contactos son determinísticos, y los enlaces de comunicación están pre-programados entre pares específicos de satélites, los contactos pueden planificarse de antemano para que se produzcan cuando dos satélites estén dentro del alcance del otro, y pueden optimizarse para minimizar las interferencias y maximizar el caudal de datos.

Ya existen soluciones de planificación, enrutamiento y toma de decisiones que fueron

diseñadas para adaptarse a sistemas distribuidos en el espacio, por lo tanto, se propone la implementación del algoritmo de enrutamiento mediante grafos de contacto (CGR, del inglés *Contact Graph Routing*) [2, 7]:, como plataforma de comunicación entre satélites con una computadora de vuelo de capacidad limitada, en una constelación con el propósito de encontrar las principales características que su implementación requiere y por ultimo conocer parámetros que midan la escalabilidad del algoritmo al aumentar los nodos de una red.

1.1. Justificación del Estudio Realizado

Satélites orbitando en formación implica mayor eficiencia para la Comisión Nacional de Actividades Espaciales CONAE a la hora de producirlos en serie, sobre todo si son satélites homogéneos, teniendo en cuenta que los costos de desarrollo disminuyen si se utilizan componentes comercialmente disponibles (COTS, del inglés *Components off The Shelf*) que ya están siendo ofrecidos y también son producidos en serie. Implica mayor robustez al no ser afectados drásticamente si uno de los elementos falla o sufre accidentes debido al ambiente espacial. En el contexto de sistemas distribuidos, CONAE propuso el proyecto Satélite de Alta Revisita (SARE), de arquitectura segmentada, que abre la posibilidad de integrar instrumentos segmentados mixtos donde pueden combinarse mediciones tomadas en distintos tiempos, en diferentes órbitas, como también simultáneamente por varios satélites orbitando uno atrás de otro en una formación tipo tren. En sistemas espaciales distribuidos, reemplazar la carga útil principal por varias cargas útiles pequeñas puede disminuir algunas limitaciones tecnológicas ya que permite el despliegue de un mayor campo de acción y aumentar la cobertura. En [8] se estudian misiones distribuidas donde mencionan como principales ventajas el aumento de la cobertura y resolución temporal [9], flexibilidad en oportunidades de lanzamiento, confiabilidad basada en la redundancia así como también la posibilidad de innovar en aplicaciones como la interferometría. En esta aplicación se aprovecha la distribución de los instrumentos para áreas donde el terreno cambia rápidamente ya que puede derivar en un cambio importante en el terreno y la reflectancia utilizando sensores distribuidos en una constelación de satélites que observen el mismo terreno desde distintos ángulos, pero sin estar excesivamente separados. En [10] se citan varios estudios al respecto. Estos ejemplos muestran la flexibilidad en los modos de adquisición o posibilidad de re-configuración en la formación geométrica de los satélites [11] en casos que se requiera.

Estos beneficios llevan consigo ciertas dificultades en cuanto a los requerimientos de alineación y posicionamiento relativo de cada satélite, la coordinación de recursos en órbita, disponibilidad del instrumento y capacidad de bajada de datos [12]. Cuando en un sistema existen instrumentos distribuidos que deben apuntar a un objetivo en específico, los satélites deben necesariamente intercambiar información para coordinar dicha observación, independientemente de que el contacto radioeléctrico con la estación terrena exista o no.

Las ventajas y desventajas de una misión de instrumentos distribuidos son consecuencia directa de la separación espacial entre los sistemas. Las desventajas son principalmente de carácter técnico debido a la complejidad operativa y procesamiento de datos, aunque gracias a la continua miniaturización de tecnologías ahora se pueden encontrar en el mercado de componentes electrónicos un gran avance en el desarrollo de microcontroladores, procesadores, memorias y sistemas embebidos que se pueden aprovechar para desarrollar software más exigente de estos recursos en computadoras de a bordo.

Aunque la coordinación de la misión, corrección de órbita y procesamiento de telemetría

se realiza típicamente en el centro de control de misión, y aprovechando que estas correcciones pueden ser realizadas en forma independiente del segmento terrestre, las distintas ventajas que nos ofrece la tecnología actual permite pensar en sistemas "inteligentes" y despierta el interés sobre inteligencia artificial para que los satélites adquieran autonomía [13], en constelaciones de satélites. La degradación de los datos debido a ruido o efectos de deriva entre periodos de contactos con las estaciones terrenas son inconvenientes comunes, y evitables mediante contactos intersatelitales directos que posibiliten la auto re-configuración en órbita [14]. La comunicación entre cada elemento de esta infraestructura garantiza su efectividad, por lo tanto se han estudiado distintos enfoques en los sistemas de redes satelitales, de los cuales, el Laboratorio de Propulsión Jet (JPL, del inglés *Jet Propulsion Laboratory*) ha contribuido con las llamadas Redes Tolerantes a Retraso en respuesta a la necesidad de comunicaciones interplanetarias.

En esta configuración cada nave espacial, satélite como también cada estación terrena puede considerarse un elemento físico o nodo dentro de la red formada mediante enlaces y cada nodo posee sus propias características como pueden ser el tipo de instrumento y su modo de adquisición, velocidad de transmisión de datos de ciencia, disponibilidad, capacidad de almacenamiento, etc.

El tamaño de cada paquete de datos y la ruta que debe seguir para llegar a destino se vuelven estructuras de datos cada vez más complejas a medida que aumentan los elementos o nodos [15], y por estas razones, es interesante investigar sobre la implementación de un sistema o algoritmo que otorgue cierta autonomía en la de toma de decisiones, que cumpla con criterios de operación en el espacio, que no sea afectado por disrupciones en la red y garantice la comunicación en constelaciones de satélites para permitir a los instrumentos trabajar con métodos de adquisición mejor coordinados. Para posibilitar esto, se plantea implementar el algoritmo CGR [7, 16], parte de un sistema de Red Superpuesta Interplanetaria (ION, del inglés *Interplanetary Overlay Network*) desarrollado por el (JPL), y estudiar el rendimiento de este algoritmo en una computadora de a bordo para servir de plataforma y dar servicio a instrumentos distribuidos en una constelación de satélites, típicamente utilizados para la observación de la Tierra, como son misiones basadas en radares y/o en cámaras ópticas, situadas en constelación u otro tipo de red con más de un satélite en visibilidad radioeléctrica.

Un punto importante a analizar es la capacidad de decisión según la necesidad de la red, es decir que, si hubiera la necesidad de configurar la constelación para que los instrumentos sean más eficientes en términos de energía, coordinando la adquisición de datos como también la transmisión de datos, el algoritmo elija la ruta que exija menos energía en la transmisión, lo que se traduce en transmitir los datos a la menor cantidad de nodos posible pero que se garantice los tiempos de retraso mínimos requeridos por el usuario. El algoritmo CGR permite este escenario si se modifican los "costos" o parámetros primarios de decisión. Esto puede ser relevante en casos donde exista una posibilidad de contacto, los nodos (satélites) pueden resultar escasos en recursos (energía, antenas, etc.) para realizarlos.

El aumento de la demanda ha contribuido al desarrollo de dispositivos disponibles comercialmente COTS y se han considerado para su uso en satélites pequeños, reduciendo significativamente los costos de desarrollo. Como resultado, las misiones *CubeSats* pueden aprovechar una ventaja económica y de tiempo de entrega significativa en comparación con los estándares de misión tradicionales.

Actualmente se cuenta con procesadores con un conjunto reducido de instrucciones (RISC, del inglés *Reduced Instruction Set Computer*) y memorias con mayor capacidad gracias a la miniaturización en componentes electrónicos. Estos cambios también han contribuido a una

nueva generación de tecnologías para pequeños satélites [4, 6, 17].

Como consecuencia, una misión *CubeSat* puede beneficiarse de una amplia gama de recursos inmediatamente disponibles que van desde componentes de satélites, subsistemas y elementos de software tanto para segmentos de vuelo como de tierra. Esto se refleja en la cantidad de *CubeSats* lanzados en los últimos años [18, 19].

Se puede inferir que el desarrollo de *hardware* en los últimos años ha incrementado la capacidad de procesamiento y almacenamiento de datos, pero desafortunadamente este desarrollo no es acompañado por los sistemas de comunicación, problema nombrado en [20], por lo que se vuelve el cuello de botella a la hora del desarrollo de aplicaciones que tienen relación con los datos generados en vuelo, ya que según [21], se necesitan importantes evoluciones tecnológicas en telecomunicaciones para satisfacer las necesidades de futuros instrumentos cuya tasa de generación de datos demandarán velocidades de descarga de 500 Mbps a 2 Gbps por cada canal de comunicación. Esto requiere la implementación de otros esquemas de modulación, novedosos sistemas de compresión de datos o directamente desarrollar nuevas maneras de manejar los recursos disponibles para la transmisión. Teniendo en cuenta estos desafíos, se decide investigar parámetros de funcionamiento del algoritmo mencionado.

La finalidad de esta implementación en una *On-board Computer*, *NanoMind* A712c, *hardware* de vuelo diseñado para microsatélites *CubeSats*, es probar la escalabilidad y conocer ciertas métricas útiles en el contexto de software, uso de memoria y la curva característica del tiempo de respuesta del algoritmo en función a la cantidad de nodos que tenga la red de satélites, con el objetivo de conocer el límite máximo en cantidad de nodos para el cual sea factible utilizar este algoritmo en computadoras limitadas, teniendo en cuenta que al aumentar la cantidad de nodos, es conveniente conocer el desempeño del procesador en cuanto al tiempo de ejecución de los algoritmos utilizados, para constelaciones de N cantidad de satélites, sabiendo que el sistema operativo también se debe encargar de otras tareas como por ejemplo el control de actitud, manejo de sensores, actuadores y comunicaciones [3].

1.2. Hipótesis

Se puede reescribir una versión de la subrutina CGR, extraída de la implementación de Red tolerante al retraso (*Delay Tolerant Network*) (DTN), ION, preparada para correr en una computadora de vuelo (OBC) de gama baja, para nano-satélites, que sea capaz de cargar los datos de un plan de contactos y encontrar las mejores rutas para la transmision de paquetes de datos en la red de satélites.

1.3. Interrogantes de la Investigación

El software para naves espaciales debe pasar por pruebas estrictas de funcionalidad para asegurar el éxito de la misión. Una de esas pruebas es el "peor escenario posible en cuanto al tiempo de ejecución" o Tiempo de Ejecución del Peor Caso (*Worst Case Execution Time*) [22].

Esta prueba de software puede determinar el tiempo de ejecución de cada tarea o función dentro del código, por lo que presenta información valiosa para la optimización del código. Es información básica para los primeros análisis de sincronización con otras tareas de control en una computadora de abordo, OBC. También se considera la tarea de coordinación en un enjambre de satélites, con pequeñas computadoras de abordo, trabajando en una red.

Durante el tiempo de vida útil del satélite son varias las tareas críticas que la computadora

de abordo OBC ejecuta. Como esta subrutina debe correr en conjunto con otras, como son los de control de actitud, control de temperatura, comandos y manejo de datos, control de la carga útil, etc., es beneficioso que todos sean optimizados y se ejecuten en el menor tiempo posible para dar tiempo a distintas interrupciones que pudieran suceder. Es por esto que se busca reducir y simplificar las subrutinas ya existentes y adaptarlas al contexto de pequeñas computadoras.

Entonces, la interrogante de la investigación es primeramente investigar el estado del arte sobre comunicaciones en constelaciones que utilizan instrumentos distribuidos y topologías en las redes que presentan para definir criterios y parámetros relevantes a la hora de una implementación de software de enrutamiento. En segundo lugar, hacer experimientos computacionales para conocer la escalabilidad (en términos de memoria y tiempo de ejecución) de un algoritmo CGR reducido, que sea una abreviación de una implementación completa de la arquitectura DTN en un software más complejo llamado ION, del cual se deba extraer y estudiar las características más relevantes y se adaptarán las subrutinas para implementarla como tarea en un sistema operativo en tiempo real FreeRTOS en una OBC comercial, propiedad de los laboratorios de integración de la Unidad de Formación Superior (UFS), la NanoMind A712C, de GomeSpace.

1.4. Objetivos

1.4.1. Objetivos Generales

- Verificar la funcionalidad del algoritmo de enrutamiento y planificación en computadoras de rendimiento limitado para dar servicio a instrumentos distribuidos en una constelación (o arquitectura distribuida) comunicados en redes tolerantes a retrasos.

1.4.2. Objetivos Específicos

- Investigar sobre el estado del arte en relación a satélites en una constelación.
- Profundizar capacidades para desarrollar software de alta complejidad en la computadora de vuelo *NanoMind* A712C.
- Implementar el algoritmo CGR en computadoras de a bordo utilizando como referencia los datos generados por un instrumento.
- Analizar variaciones de métricas que minimicen el uso de energía total para la transmisión de datos (mínima cantidad de saltos en la red) y estudiar su escalabilidad en términos de tiempo de ejecución y uso de memoria.
- Integrar la implementación con software de vuelo de código abierto.

CAPÍTULO 2

Estado del Arte

En los párrafos siguientes se pretende organizar los puntos estudiados hasta llegar a esta propuesta de investigación. Primeramente se estudiaron los sistemas de instrumentos distribuidos teniendo en cuenta el desarrollo del TRONADOR II y III y el interés de CONAE en la puesta en órbita de las cargas livianas como instrumentos segmentados. Para delimitar el alcance, se decide hacer un aporte desde el punto de vista de las comunicaciones satelitales por medio de las redes tolerantes a retrasos como manera de garantizar la coordinación de un sistema de satélites en una constelación y dentro de ese contexto implementar el algoritmo CGR, y utilizar como generador de datos de referencia un instrumento óptico.

Las misiones espaciales que implican sistemas de satélites distribuidos que utilizan distintos instrumentos para observación terrestre se han vuelto interesantes en las últimas décadas debido a las numerosas ventajas que presentan.

El concepto de misiones espaciales distribuidas fue primeramente introducido en aplicaciones astronómicas y planetarias, pero últimamente ese enfoque también fue propuesto para observación de la Tierra, ejemplo actual de esto es la misión SAOCOM 1-A y SAOCOM 1-B, que llevan como carga útil un instrumento de radar polarimétrico, con una antena de 35 metros cuadrados y ya forman parte de la constelación por la necesidad de obtener los intervalos de revisita requeridos por los usuarios. Estos dos satélites en conjunto con cuatro satélites COSMO-SkyMed equipados con tecnología SAR de Banda X de la ASI, conforman la constelación Sistema Italo Argentino de Satélites para la Gestión de Emergencias (SIASGE).

Esta tendencia ha llevado a varios investigadores de distintos campos de la ciencia a relacionar sus investigaciones con el campo espacial. Los primeros estudios datan de la década de 1980, y varios trabajos, mayormente sobre dinámica fueron publicados en 1990 y posteriormente puestos a prueba en misiones como GRACE[23], PRISMA[24] y Tandem-X[25], que fueron los precursores de las futuras misiones distribuidas. Los objetivos principales de la misión PRISMA son realizar demostraciones de experimentos de guiado, navegación y control para una familia de futuras misiones donde se planea utilizar satélites en formación. En esta misión, los instrumentos a bordo son dos receptores GPS, cabe resaltar que son componentes basados en COTS y la bajada de datos de ciencia, además de realizarse mediante un enlace directo a tierra en Banda S, también se descarga mediante un enlace inter-satelital en UHF. El proyecto de SAR Biestático TerraSAR-X/F-SAR fue el pionero en misiones radar SAR distribuidas y su importancia radica en que sirvió de prueba para algoritmos de sincronización, calibración y captura de imágenes avanzadas. Siendo todo esto necesario debido a la operación independiente del receptor y el emisor. Los resultados indican que en comparación a la operación de un satélite monolítico, el SAR Biestático obtuvo mejores resultados en cuanto a resolución espacial y relación señal a ruido. Estos resultados son explicados en [26]. Varias misiones de observación de la tierra son valoradas internacionalmente, ejemplo de esto es el COSMO-SkyMed, el Monitoreo global del Medio Ambiente y la Seguridad (*Global Monitoring for Environment and Security*) (GMES) que contribuye con datos satelitales a

nivel internacional. Para esto los satélites (Sentinel-1 hasta Sentinel-5) necesitan cooperar con otras misiones de la ESA, o de otras agencias estatales o privadas con la finalidad de garantizar el éxito de monitoreo de desastres a nivel global. "Las nuevas misiones de Sentinel proporcionan información para todos los dominios de GMES, comenzando con las imágenes de radar diurno y nocturno de todo el clima de Sentinel-1 para servicios terrestres y oceánicos. Sentinel-2 entregará imágenes ópticas de alta resolución para servicios terrestres y Sentinel-3 brindará será útil para el monitoreo de la tierra en el océano y en el mundo. Sentinel-4 y Sentinel-5 proporcionarán datos para el monitoreo de la composición atmosférica desde las órbitas geoestacionarias y polares, respectivamente" [27]. También se puede citar misiones relevantes como *Iridium Next*, utilizada para servicios de seguridad en tierra, mar y aire con capacidad de seguimiento de aeronaves y *LEOSat*, la constelación de satélites de comunicaciones de alta velocidad que utiliza instrumentos de comunicación láser (ópticos).

2.1. Comunicaciones Satelitales

Se entiende por comunicaciones satelitales a la transferencia de datos realizadas a través de satélites y se dan mediante la transmisión y recepción de señales electromagnéticas moduladas para empaquetar y enviar datos, de un nodo transmisor a otro receptor, en el cual el satélite recibe la señal, la procesa, y luego trasmite la señal deseada.

Las comunicaciones satelitales ofrecen características únicas en comparación a las tecnologías de transmisión terrestre, como costo independiente a la distancia, costo de transmisión independiente a la cantidad de nodos, baja tasa de error, alta capacidad de canal, y servicio a diversas redes de usuarios [28].

El sistema de comunicaciones es el eslabón final en la cadena de manejo y procesamiento de datos en satélites de observación de la tierra, misiones de exploración de espacio profundo, y carga útil principal en satélites de comunicaciones.

2.1.1. Historia

La primera referencia conocida sobre satélites geoestacionarios es la carta de Arthur C. Clark, titulada *"Peacetime Uses for V2"*, al editor de la revista Wireless World en 1945 [29], Los satélites artificiales fueron propuestos como relés de comunicaciones en [30], donde un aparato en órbita ecuatorial, con un radio orbital de 42000 km a una velocidad que permita estar en sincronía con la tierra sea capaz de retransmitir las ondas de radio desde cualquier punto de la tierra que este en la linea de visión del satélite para garantizar comunicaciones globales.

Este artículo pasó desapercibido hasta que en 1965 se convirtió en realidad con el lanzamiento del *Intelsat I*, el primer satélite de comunicaciones geoestacionario comercial.

A una altitud de 35.786,04 kilómetros, que permite que el período orbital del satélite coincida exactamente con el de la rotación de la Tierra. En esta órbita y mediante correcciones periódicas en su velocidad, quedan inmóviles, o estacionarios, relativos a un observador en la Tierra, por lo que se los llama satélites geosíncronos. Esta característica permite el uso de antenas fijas, debido a que su apuntamiento no necesita cambiar y garantiza el permanente contacto con el satélite.

La industria de comunicaciones satelitales empezó a mitad de la década de 1960 y se popularizó gracias a la demanda de sistemas de transmisión de programas televisivos, datos en

redes privadas o militares, comunicaciones móviles, etc. Actualmente es ha vuelto parte de la vida cotidiana al verse antenas en forma de platos para la recepción de señales de televisión.

En las últimas décadas las comunicaciones intersatelitales han tomado gran relevancia por las necesidades de conectividad que surgen con el crecimiento del mercado a nivel mundial. El desarrollo de internet terrestre ha permitido un avance tecnológico histórico en las comunicaciones a pesar de que el acceso a la red no es del todo global actualmente. En este sentido los satélites han logrado salvar, por así decirlo, la necesidad de comunicación en cualquier parte del mundo mediante satélites de órbita geosíncrona.

Servicios de información como televisión e internet o inclusive telefonía pueden ser ofrecidos gracias a satélites de comunicación y misiones como *INMARSAT, Iridium, Thuraya, Globastar.*

Estos servicios son de una constelación para cobertura global constante, de una cantidad considerable de satélites. Pero el montaje de este tipo de redes implica costos sumamente elevados.

Actualmente se busca brindar servicios similares de información y comunicación en regiones, o países en vías de desarrollo o lugares de escasa población, pero mediante satélites en órbita baja para aplicaciones que no se necesiten una conexión permanente entre el emisor y el receptor [31], [32].

Los satélites de órbita baja LEO, están localizados a una distancia reducida con relación a la Tierra (160-2.000 km) y ofrecen la ventaja de reducir la pérdida por propagación y el retardo, debido a que existe una distancia menor para la propagación de las señales. Estos satélites poseen un movimiento relativo a la Tierra para poder mantenerse en órbita (con un período orbital típico de alrededor de 90-100 minutos), esto representa una ventaja para la observación de la Tierra, y Redes Tolerantes a Retraso DTN, dado que toda la Tierra puede ser explorada por un solo satélite, disminuyendo así los costos, y el satélite además de observar la tierra puede utilizarse como relay de comunicaciones aprovechando que explora toda la Tierra y puede hacer contacto varios nodos en tierra (los satélites geo-síncronos generalmente observan y tienen contacto con solo una parte de la superficie) pero requiere constelaciones de decenas de satélites para una cobertura global contínua similar a un servicio de telefonía o acceso a Internet.

En la actualidad *SpaceX* ha anunciado el programa *Starlink* en el cuál prevén el lanzamiento de una constelación de 40.000 satélites LEO para proveer servicios de Internet a escala global sin la desventaja que implica la latencia inherente de satélites en órbita geo-síncrona.

Desde que comenzó a lanzar satélites de *Starlink* en 2019, *SpaceX* ha lanzado más de 1.500 satélites al espacio. Según la información pública disponible, hasta el 13 de abril de 2023, *SpaceX* ha lanzado un total de 1,853 satélites para su constelación de satélites *Starlink*. Estos satélites están diseñados para orbitar a una altitud de alrededor de 550 kilómetros sobre la Tierra, mucho más cerca que los satélites de comunicaciones convencionales que suelen orbitar a altitudes de varios miles de kilómetros.

Este proyecto es netamente comercial y está pensado teniendo en cuenta el mercado, pues cerca de la mitad de la población mundial no tiene acceso regular a Internet.

El peso promedio de los satélites de *Starlink* es de alrededor de 260 kg (573 libras) por lo que están dentro de la categoría de mini-satélites. Cada satélite individual está diseñado para ser compacto y eficiente en términos de peso, lo que permite a *SpaceX* lanzar múltiples satélites en cada cohete Falcon 9.

Estos satélites tienen una masa seca (sin combustible) de alrededor de 227 kg (500 libras) y un tanque de combustible lleno puede aumentar el peso total del satélite a alrededor de 260 kg (573 libras). Este diseño de bajo peso ayuda a reducir los costos de lanzamiento y permite

a *SpaceX* lanzar y desplegar grandes constelaciones de satélites con eficiencia y regularidad.En términos de parámetros de latencia, los documentos técnicos de *SpaceX* indican que la empresa espera que la latencia de su red de satélites de *Starlink* sea de alrededor de 20 a 40 milisegundos (ms), lo que representa una reducción significativa en comparación con las redes de satélites de comunicaciones convencionales que suelen tener latencias de cientos de milisegundos [33].

Aunque el sistema de satélites de órbita baja de *Starlink* ha generado algunas preocupaciones en torno a la cantidad de desechos espaciales que puede generar, la empresa ha tomado medidas para mitigar estos riesgos, como implementar sistemas de maniobra de satélites para evitar colisiones y planificar la eliminación controlada de satélites al final de su vida útil.

Otra compañía y principal competencia de *SpaceX* que tiene en mente este tipo de negocios es *OneWeb* y planea el lanzamiento de 800 satélites LEO. Hasta el 13 de abril de 2023, la compañía OneWeb ha lanzado un total de 322 satélites para su constelación de satélites de banda ancha de baja órbita terrestre.

OneWeb lanzó su primer lote de satélites en febrero de 2019, y desde entonces ha llevado a cabo varios lanzamientos adicionales, con el objetivo de crear una constelación de satélites de banda ancha de baja órbita terrestre para proporcionar servicios de internet globales de alta velocidad y baja latencia.

En noviembre de 2020, *OneWeb* completó su séptimo lanzamiento de satélites, llevando su constelación a un total de 74 satélites en órbita. A partir de ese momento, *OneWeb* continuó con su plan de despliegue de satélites para completar la constelación, con el objetivo de proporcionar servicios de internet a los usuarios finales en el futuro cercano.

A pesar de los reveses causados por la pandemia de COVID-19 y la declaración de bancarrota en marzo de 2020, *OneWeb* ha continuado trabajando en el desarrollo y despliegue de su red de satélites de banda ancha. En noviembre de 2021, *OneWeb* anunció que había completado su constelación de satélites, con un total de 648 satélites en órbita.

Aunque aún no se ha anunciado una fecha oficial de lanzamiento para el servicio de internet de *OneWeb*, se espera que la compañía comience a ofrecer servicios de internet en algunos mercados seleccionados a partir de finales de 2021 y principios de 2022, con planes de expandirse a nivel mundial en los próximos años [34].

Una diferencia importante es el tipo de frecuencia utilizada para las comunicaciones satelitales. Mientras que *Starlink* utiliza principalmente frecuencias de banda Ku y Ka para sus servicios, *OneWeb* utiliza principalmente frecuencias de banda L. La banda L se considera mejor para la cobertura global, ya que las señales pueden penetrar mejor a través de la atmósfera terrestre y las condiciones meteorológicas adversas [34].

También existen diferencias en la arquitectura de la red y en la tecnología utilizada para proporcionar servicios de internet. *Starlink* utiliza una arquitectura de red distribuida con enrutamiento dinámico y procesamiento de señales digitales avanzado, mientras que *OneWeb* utiliza una arquitectura de red más centralizada y utiliza tecnología de enrutamiento de paquetes. Este sistema de enrutamiento utiliza una serie de algoritmos de enrutamiento que permiten a los satélites de *OneWeb* seleccionar la mejor ruta para enviar los paquetes de datos a través de la red.

El sistema de enrutamiento de paquetes de *OneWeb* se basa en una arquitectura de red centralizada, donde los satélites se comunican con un centro de control terrestre para recibir instrucciones de enrutamiento y configuración de red. El centro de control terrestre es responsable de monitorear el estado de la red y ajustar los parámetros de enrutamiento para optimizar el rendimiento y la eficiencia de la red.

Por otro lado, está siendo desarrollada una red de sensores en el espacio para demostración de avances tecnológicos como computación distribuida a bordo, procesamiento de señales en dispositivos reconfigurables, ESPACENET [35], que pueden ser aplicadas a misiones de sensado remoto multipunto, procesamiento de imágenes colaborativo, comunicación continua y mantenimiento de órbita [36]. También en [37] los autores caracterizan 9 misiones interesantes donde se resaltan los experimentos realizados con satélites en formación y misiones de pequeños satélites múltiples.

Al tener esta cantidad de satélites o nodos, estas empresas y misiones tecnológicas necesitan conocer sobre redes de satélites y se hace interesante estudiar las características de los algoritmos de enrutamiento de paquetes de datos en una red, por lo que en los párrafos siguientes se describen formas innovadoras de lidiar con los problemas de desconexión o conexión interrumpida o intermitente, con el propósito de utilizar estos métodos como plataforma de servicios de instrumentos distribuidos.

2.2. Desafíos en las Comunicaciones Satelitales

Un satélite de observación de la Tierra está equipado con una variedad de instrumentos para capturar datos de la superficie terrestre, la atmósfera, el océano y otros fenómenos relacionados con el medio ambiente. Se pueden distribuir estos instrumentos en una constelación de satélites para mejorar la cobertura y la frecuencia de muestreo de los datos recopilados. Al utilizar una constelación de satélites, se pueden recopilar datos de diferentes partes de la Tierra al mismo tiempo, lo que aumenta la cobertura global y reduce los tiempos de revisita.

Además, al tener múltiples satélites que llevan diferentes tipos de instrumentos, se pueden recopilar diferentes tipos de datos para complementar y mejorar la calidad y precisión de los resultados. Por ejemplo, un satélite puede llevar una cámara de alta resolución para capturar imágenes de alta calidad de la superficie terrestre, mientras que otro satélite puede llevar un radar para medir la elevación de la superficie terrestre y detectar cambios topográficos.

La constelación de satélites también puede permitir la distribución de la carga de trabajo entre los satélites, lo que puede mejorar la eficiencia de la misión y prolongar la vida útil de los satélites individuales. En resumen, una constelación de satélites puede mejorar significativamente la capacidad de observación de la Tierra y proporcionar datos más completos y precisos para su análisis y aplicación.

Pero estos beneficios necesitan de la construcción de una estructura de red que sea capaz de conectar redes en tierra, redes oceánicas, redes de órbita baja y de espacio profundo, y al mismo tiempo ofrecer un servicio de conexión extendida, confiabilidad apreciable por medio de múltiples rutas además de recursos y servicios compartidos en órbita, obliga a tener en cuenta varios detalles.

Las misiones distribuidas se pueden clasificar en dos categorías distintas. (a) La reducción de datos basada en tierra requiere que todos los datos científicos se transfieran a tierra a través de una capacidad de enlace espacio-tierra independiente asociada a cada miembro (nodo) de la red. (b) La reducción de datos desde el espacio requiere que una o más naves espaciales recojan y procesen los datos de los otros miembros de la distribución. Esta alternativa requiere enlaces entre satélites para transferir datos científicos entre los miembros. En tales circunstancias, los datos podrían ser procesados en su totalidad, parcialmente o simplemente re-empaquetados antes de ser transferidos al segmento terreno de la misión. La elección de las alternativas de transferencia de datos, es decir, el uso de enlaces entre satélites, depende del diseño y los

objetivos de la misión.

Para esta estructura, se deben considerar aspectos del medio en el que se desempeñan.

En tierra estos aspectos están siendo superados mediante conexiones por conductores eléctricos, radiofrecuencia, y tendidos de fibra óptica intercontinental. Pero en el espacio se presentan desafíos como la dinámica orbital, cambios constantes en las distancias entre nodos, la misma rotación de la tierra, situaciones fortuitas en el satélite asociados a los SEL, y las limitaciones de energía [38].

En relación a las limitaciones de memoria, en DTN para pequeños satélites no se considera debido a que el tamaño de datos generados por los instrumentos (en pequeños satélites) no son masivos en comparación a la capacidad de almacenamiento que pueden tener abordo, el cuello de botella generalmente se da en la transmisión de los datos a tierra, utilizando enlaces UHF/VHF y Banda S. Aunque ya existen estudios de optimización del rendimiento de memoria en DTN para espacio profundo [39], donde, las simulaciones muestran que sin afectar la correcta entrega de los datos y con el fin de obtener una mejor estrategia de asignación de recursos de memoria, se proponen como esquema de optimización conjunta la longitud optimizada del segmento de datos LTP y el número optimizado de paquetes agregados por cada bloque LTP.

Obviando las dificultades citadas, las comunicaciones por satélite presentan beneficios tecnológicos cuando los enlaces terrestres quedan inhabilitados como en caso de desastres naturales, incendios, inundaciones o terremotos [40], sin mencionar los aspectos monetarios derivados de aplicaciones como la observación de la Tierra para monitoreo de cultivos y producción en general.

Al ser utilizada como línea de emergencia, el sistema debe presentar robustez, confiabilidad, además de tener prestaciones similares a las redes en tierra. Para esto se debe considerar soluciones en cada una de las capas de red.

En particular, si nos centramos en la capa de transporte y las capas superiores, el rendimiento se ve desafiado por largos tiempos de ida y vuelta (RTT, del inglés *Round Time Trip*), especialmente para los sistemas (GEO) (aproximadamente 600 ms) que surgen en el entorno espacial, como la distancia de separación y el movimiento relativo entre dos estaciones receptoras/transmisoras para los sistemas LEO.

Una posible solución propuesta por los autores en [40] es modificar la capa de transporte para adaptarse a estas incertidumbres, con el fin de aplicar variantes de protocolo de transporte adecuado para el enlace por satélite.

A pesar de estos inconvenientes, el avance de la tecnología ofrece múltiples herramientas para distintos propósitos. Actualmente, los procesadores ARM esta desplazando a otros fabricantes del mercado y se está volviendo líder en este segmento debido a sus constantes avances en poder de procesamiento y la ventaja de menor consumo de energía [41]. El modelo de negocios de ARM también es disruptivo. Diseña los núcleos y licencia la propiedad intelectual a organizaciones que se encargan de construir circuitos integrados de acuerdo a los requerimientos del usuario final que pueden agregar funciones como procesador digital de señales, circuitos integrados reconfigurables (FPGA, del inglés Field Programable Gate Arrays), las memorias de alto rendimiento y alta tasa de transferencia, posibilitan el desarrollo de software que flexibiliza y facilita lidiar con estos desafíos.

2.2.1. Utilización de COTS en Satélites

Este avance tecnológico permitió la estandarización de componentes especializados para el entorno espacial, los llamados *CubeSats* ahora pueden montarse o desarrollarse en un tiempo

relativamente corto en comparación a satélites más tradicionales. Ahora los dispositivos COTS pueden adquirirse y luego integrarse para aplicaciones espaciales, aunque esto trae consigo otra lista de desafíos que se debe considerar.

Muchos COTS pueden utilizarse para misiones satelitales pero pueden comprometer parámetros de confiabilidad y accesibilidad. Generalmente los fabricantes realizan misiones de pruebas tecnológicas para asegurarse de que sus dispositivos son capaces de soportar las condiciones del entorno espacial.

Según [17], el 30 % de los CubeSats fallan a los 20 días del lanzamiento, siendo la OBC uno de los subsistemas que más fallos causan, junto con el subsistema de distribución de la EPS y el subsistema de comunicaciones COM. Estos datos muestran la necesidad y la criticidad de las pruebas en tierra. Las misiones con extensos ciclos de vida, como las misiones a espacio profundo, suponen un mayor nivel de dificultad, debido a la radiación que los dispositivos deben soportar [42].

Tradicionalmente los procesadores de grado espacial utilizaban un sólo núcleo [43], consecuencia de estar endurecidos para soportar la radiación, y actualmente el mercado ofrece procesadores de arquitectura avanzada, con varios núcleos, en plataformas que incluyen DSP y FPGA. Empresas como *NanoAvionics* ofrecen procesadores basados en la familia STM32, *GomSpace* ofrece la *NanoMind* A712, todos a su vez basados en ARM.

Para avanzar en el desarrollo de aplicaciones de mayor complejidad, es de utilidad conocer las ventajas características de los procesadores actuales (los ARM) cuando se corren en ellos algoritmos de enrutamiento, para hacer una elección educada en los procesos de adquisición.

Procesadores

Los procesadores para aplicaciones espaciales deben garantizar el funcionamiento en condiciones adversas, propias del ambiente espacial. Para asegurar el éxito de la misión es primordial conocer las características de los procesadores en las OBC a utilizar.

Para el efecto, tradicionalmente se utilizan procesadores que son fabricados en procesos dedicados de endurecimiento a la radiación (RH, del inglés *Radiation Hardened*) que son capaces de soportar niveles de (TID, del inglés *Total Ionizing Dose*): 100-300 kRad para órbitas GEO y 10-50 kRad para LEO, como también *Latch-up's*, SEU, SEL, SEFI. Algunos procesadores conocidos son los *Harris RH3000* 32-bit, *RAD6000*, basado en la arquitectura IBM System/6000, utilizado en el *Mars Rover*, y su actual versión, el RAD750 que fue liberado para la venta [44], de donde se desprende el *RAD5545® SpaceVPX Multi-Core Single Board Computer* [45]. Estos componentes tienen la desventaja de ser costosos debido al bajo volumen de producción, un nicho de mercado pequeño y los costos de calificación.

Seguidamente existen los procesadores que son diseñados tipo (ASIC, del inglés Application-Specific Integrated Circuit) y fabricados en procesos comerciales, donde el endurecimiento a la radiación está dada por diseño, de donde viene el nombre (RHBD, del inglés *Radiation Hardened by Design*). Son menos costosos que los RH, pero más que los COTS, debido a su fabricación menos complicada. Algunos procesadores conocidos de este tipo están basados en la arquitectura LEON, de origen europeo. El SPARC V7 ERC32 y TSC695FL, fabricados en ATMEL, Francia.

En la actualidad se pueden encontrar CPU's, PIC, MSP, AVR32 de ATMEL, ARM que presentan diversas soluciones y opciones de funcionamiento en el mercado de los procesadores de bajo costo. Las más utilizadas son las ARM® Cortex M3, ARM CortexTM M7, LEON3FT. En [42] se puede encontrar un estudio del rendimiento de distintos procesadores.

Sistemas Operativos

Los sistemas operativos utilizados para aplicaciones en el espacio deben mínimamente dar soporte a un conjunto de herramientas de *hardware*, en tiempo real, manejando varios protocolos de comunicación, en diferentes arquitecturas y utilizar opciones de manejo eficiente de energía. El manejo de dinámico de memoria RAM es crítico ya que en implementaciones de lenguaje C, usualmente son basadas en el tiempo y por lo tanto no son determinísticos, lo que puede suponer un quiebre en la ejecución de las aplicaciones.

Existen diversos sistemas operativos utilizados en el sector aeroespacial que han sido y siguen siendo las primeras opciones. *VxWorks* fue desarrollado inicialmente en 1987 por *Wind River* (que ahora es propiedad de *Intel*), *VxWorks* es un núcleo monolítico que admite principalmente plataformas ARM e Intel. Por otro lado, esta *FreeRTOS*, desarrollado originalmente por *Richard Barry* en 2002, mantenido y distribuido por *Real Time Engineers Ltd*. *FreeRTOS* se implementa en varios entornos industriales/comerciales y es la base de varios proyectos de investigación. A diferencia de muchos otros RTOS, *FreeRTOS* está diseñado para ser pequeño, simple, portátil y fácil de usar. Es, determinístico, multi-hilo, apropiativo [46] y utiliza una arquitectura con microkernel. Por lo tanto, cuenta con el respaldo de una gran comunidad y se ha portado a una gran cantidad de MCU, incluido el hardware disponible en el banco de pruebas abierto. Una interesante recopilación de sistemas operativos para pequeños dispositivos embebidos se puede encontrar en [47], aunque en este trabajo se realiza la implementación del algoritmo en el sistema operativo FreeRTOS, el cual es un Sistema Operativo en Tiempo Real de código libre, determinístico, multi-hilo, apropiativo [48].

2.3. Redes Terrestres TCP/IP

En estas redes, la principal característica relevante a esta investigación es que suponen conectividad continua y retrasos cortos, los enrutadores realizan el almacenamiento no persistentes (a corto plazo), la información se almacena persistentemente solo en los nodos finales, es decir, fuera de la red. Esto se debe a que, al tratarse de una transmisión confiable, se supone que la información se puede recuperar fácil y directamente de la fuente [40].

TCP es, en términos del modelo OSI, un protocolo de capa de transporte. Proporciona una conexión de circuito virtual confiable entre aplicaciones; es decir, se establece una conexión antes de que comience la transmisión de datos. Antes de que pueda ocurrir cualquier característica de TCP, primero se debe establecer la conexión TCP, porque TCP es un protocolo orientado a la conexión.

Un protocolo orientado a la conexión es un protocolo que establece una conexión permanente entre el cliente y el servidor antes de que pueda comenzar la transferencia de datos. Durante el establecimiento de esta conexión, un dispositivo negocia la cantidad de tráfico que se reenviará durante el protocolo de enlace de tres vías, que debe completarse antes de que pueda comenzar la transferencia de datos.

Los datos se envían sin errores ni duplicados y se reciben en el mismo orden en que se envían. No se imponen límites a los datos; TCP trata los datos como un flujo de bytes.

Un flujo continuo de bytes se divide en segmentos para su transmisión y entrega por parte de los servicios de la capa de transporte. La mayoría de las redes tienen una limitación en la cantidad de datos que puede contener un solo paquete. Debido a esto, la capa de transporte del dispositivo de envío prepara los datos en segmentos. De manera similar, la capa de transporte del dispositivo receptor recibe estos segmentos y usa el encabezado para reconstruirlos en datos

completos [49] En términos del modelo OSI, IP es un protocolo de capa de red. Proporciona un servicio de datagramas entre aplicaciones, compatible con TCP y UDP.

La retransmisión de paquetes de datos en una red basada en DTN se diferencia en varios aspectos de la transmisión de paquetes a través de una red basada en IP.

El principal objetivo del diseño de TCP/IP era construir una interconexión de redes, conocida como internetwork, o internet, que proporcionaba servicios de comunicación universales a través de redes físicas heterogéneas. El claro beneficio de tal internetwork es la habilitación de la comunicación entre servidores en diferentes redes, quizás separadas por una gran área geográfica.

Al enviar datos a un destino remoto, un servidor pasa datagramas a un enrutador local, el enrutador reenvía los datagramas hacia el destino final. Viajan de un enrutador a otro hasta que llegan a un enrutador conectado al segmento LAN del destino. Cada enrutador a lo largo de la ruta de un extremo a otro selecciona el dispositivo del siguiente salto que se usa para llegar al destino. El siguiente salto representa el siguiente dispositivo a lo largo de la ruta para llegar al destino. Está ubicado en una red física conectada a este sistema intermedio. Debido a que esta red física difiere de aquella en la que el sistema recibió originalmente el datagrama, el host intermedio ha reenviado (es decir, "enrutado") el datagrama IP de una red física a otra.

En el protocolo IP, el enrutamiento esta determinado por la estructura de conectividad actual, que no sufre cambios intermitentes. La ruta a un destino dado, una vez calculada, puede ser almacenada en una tabla de enrutamiento para referencia y será actualizada cuando cambien los parámetros de conectividad. Esta es transmitida mediante mensajes propios del protocolo que son generados inmediatamente en respuesta a cambios detectados en la conectividad que invalida esa ruta. Comparando este procedimiento con el enrutamiento en DTN, se diferencia en que el potencial retraso en la llegada de la información, relativa a los cambios de conectividad, hace que toda esta información sea potencialmente obsoleta; un nodo que se basó únicamente en este flujo de información nunca puede tener una comprensión totalmente exacta de la conectividad actual en la red [50].

2.4. Redes Satelitales

Los sistemas de adquisición y transmisión de datos implican un desafío tecnológico en la topología de una red por la complejidad y robustez necesaria en la comunicación inter-satelital. Debido al movimiento relativo y a las características adversas del ambiente espacial, la demora o disrupción son inevitables cuando satélites vecinos necesitan intercambiar información. La tasa de transferencia de datos puede ser insuficiente para el tiempo en el que dos satélites pueden establecer el enlace, o sencillamente pueden existir obstrucciones en el canal. En una constelación de satélites, cuando sea necesario coordinar la formación de vuelo, modos de adquisición de imágenes o datos en general, ignorar tales inconvenientes puede afectar negativamente el rendimiento del control y hacer impredecible la estabilidad del sistema [51]. Para manejar esta incertidumbre se deben adoptar topologías de red sumamente robustas o novedosos protocolos que minimicen los riesgos.

En constelaciones de satélites todas las comunicaciones y coordinaciones son canalizadas a través de la estación terrena por razones de seguridad. La degradación de los datos de observación por efectos del ruido o efectos de deriva entre periodos de contacto con la estación terrena son inconvenientes que pueden ser evitados mediante enlaces intersatelitales, por lo tanto, son deseables para la auto-reconfiguración en órbita de la constelación [14].

Las arquitecturas de red centralizadas resultan susceptibles a fallas del nodo central o punto de acceso (AP, del inglés *Access Point*) por lo que este debe ser continuamente monitoreado, además de ser ineficiente para comunicación entre satélites vecinos. Como el consumo de energía aumenta exponencialmente para una extensión lineal de la comunicación, la eficiencia energética de dichos sistemas centralizados es baja [52, pág. 347].

En el enfoque descentralizado (como las redes ad-hoc), no se necesita un coordinador de acceso al medio y tampoco se utiliza una confirmación de que el paquete llegó a destino, lo que conlleva a reducir los requisitos de hardware, pero causa un mayor esfuerzo de desarrollo de software. Soluciones terrestres están bien implementadas desde hace décadas (TCP/IP), pero todavía tienen que ser adaptados al uso en el espacio [52, pág 349].

Los resultados de los experimentos de la misión satelital UWE-1 mostrados [53], son que la comunicación IP es posible, pero son necesarias varias modificaciones en cuanto a optimización para permitir una eficiente comunicación entre satélites y estaciones terrestres. Especialmente la alta tasa de error de paquete (PER, del inglés *Packet Error Rate*) observada en el enlace de comunicación con UWE-1 afecta al rendimiento del protocolo AX.25. Entonces se necesita desarrollar nuevos mecanismos para el mejoramiento de la aplicación de dicho protocolo.

En [14, pág. 355] se presenta el concepto de segmento espacial distribuido donde varias estaciones terrenas distribuidas contribuyen al flujo de datos. El ejemplo presentado es de la misión QB50 donde 50 pico-satélites idénticos son provistos por 50 instituciones diferentes que cooperan para proveer datos de ciencia [54].

2.4.1. Redes Tolerantes a Retrasos (DTN)

Las redes DTN proponen una solución general a los problemas de largos retrasos, conectividad intermitente, asimetría, y altas tasas de error usando un servicio de almacenamiento y retransmisión para garantizar el flujo de datos de un nodo fuente a otro nodo destino de la red, manejando además datos operativos referentes a los intervalos de conectividad en descriptos en función del tiempo, reportes de estado del paquete, advertencias y diagnóstico [55].

El concepto de una red tolerante a retraso es una red abstracta que no es especificada para una capa de red en particular. La tolerancia a los retrasos implica que es posible la comunicación entre los nodos (receptores/transmisores de paquetes), sin que la red entre en conflictos [56].

Debido a la necesidad de garantizar las comunicaciones en escenarios donde existen interrupciones intermitentes en el canal, pueden tener como principales objetivos maximizar la probabilidad de entrega del mensaje, minimizar la latencia de extremo a extremo, como también minimizar la cantidad de "saltos" que da el paquete hasta llegar a destino [57].

Las redes DTN fueron diseñadas para nodos sin conectividad de red continua [15, 16]. Como una posible solución que resuelvan los problemas de nodos ocasionalmente conectados en el espacio, donde las interrupciones del enlace debido a una distancia de visibilidad radioeléctrica limitada o debido a efectos de ocultamiento o simplemente perdidas de enlace, como resultado de que las distancias cambian dinámicamente en las diversas órbitas satelitales.

Como cuando una nave espacial está enviando información desde el espacio a la tierra o viceversa, pierde conectividad debido a que pasa detrás de un planeta o cualquier otro tipo de obstrucción y pierde visibilidad radioeléctrica, que imposibilitan el uso de protocolos de red tradicionales, las redes tolerantes a retrasos (DTN, del inglés *Delay Tolerant Network*) surgen como propuestas de adaptación a estas circunstancias, donde en el momento de la desconexión, otra nave espacial puede transportar el paquete de datos y retransmitirla a otro nodo, en un

Tabla 2.1 – *Elementos de datos del paquete (Relevante para el enrutamiento)*

Parámetros de bloque primarios del paquete	
$B.src$ & $B.dst$	Nodos Fuente y Destino para el paquete
$B.size$	Tamaño del paquete, incluyendo cabecera y carga útil
$B.p$ or $priority$	Clase de prioridad del paquete (1... p)
$B.critical$	Bandera de criticidad del paquete
$B.custody$	Bandera requerida de transferencia de custodia
$B.fragment$	Bandera de autorización de fragmentación
$B.deadline$	Tiempo de Expiración del paquete
Parámetros computados	
$B.EVC$	Consumo de volumen estimado ($size * 1{,}03$) [59]
$B.sender$	Remitente previo del paquete, informado por CLA

proceso sucesivo, hasta hacer llegar este paquete a destino, estableciendo la infraestructura de una red.

Por lo tanto, las redes DTN han atraído interés en la comunidad espacial [23, 24].

El estándar utilizado para desarrollar la red DTN ha sido el protocolo llamado *Bundle Protocol* (BP), o protocolo de paquetes, donde cada *bundle* es una unidad de dato del DTN *Bundle Protocol* y cada nodo o *Bundle Node* dentro de la red es capaz de enviar o recibir paquetes. Cada nodo posee tres componentes conceptuales: *Bundle Protocol Agent* (BPA), *Convergence Layer Adapters* (CLA), *Application Agent*, cuyos detalles son descriptos en [16].

Las redes DTN, se ha desarrollado bastante en la última década según [32], donde se evidencia la mejoría en la utilización de recursos y menor tasa de fallos en los diferentes casos de satélites orbitando en formación. En las redes DTN se aplica un concepto de "almacenamiento y retransmisión", donde los paquetes de datos se almacenan en cada nodo y se pasan los datos consecutivamente a los nodos disponibles en la esperanza de alcanzar finalmente el nodo de destino.

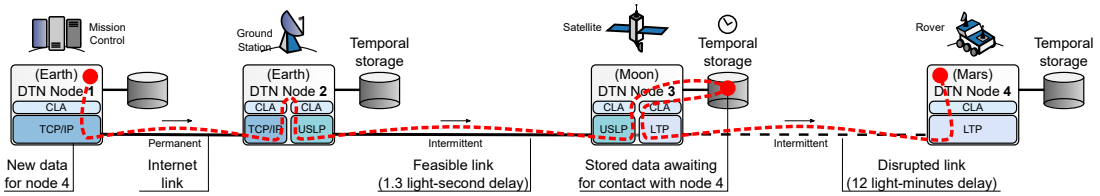
Para tales situaciones, primero se necesita calcular o establecer una ruta completa que el paquete de datos debe seguir y, posteriormente, se transfieren los datos. Las rutas son entendidas como el esquema ordenado de retransmisión consecutiva de datos, de un nodo a otro, hasta llegar a destino. Estas rutas son calculadas en cada nodo, teniendo disponible información sobre los enlaces entre los nodos en una escala temporal, llamada "plan de contactos", que fue cuidadosamente calculada mediante propagadores orbitales y luego distribuida a toda la red.

2.5. Aspectos Generales

2.5.1. Paquete (Bundle)

La definición del concepto de un paquete está dada en [58] como:

Un paquete es una unidad de datos de protocolo del "DTN *Bundle Protocol*". Cada paquete comprende una secuencia de dos o más "bloques" de datos de protocolo, que sirven para la ejecución de algoritmos de selección de rutas 2.1. Varias instancias del mismo paquete (la misma unidad de datos del protocolo DTN) pueden existir simultáneamente en diferentes partes de una red, posiblemente en diferentes representaciones, en la memoria local de uno o más nodos del paquete y/o en tránsito entre nodos. En el contexto de la operación de un nodo de paquete, un paquete es una instancia de algún paquete en la red que se encuentra en la memoria local de ese nodo.



	Mercury	Venus	Earth	Mars	Jupiter	Saturn	Uranus	Neptune
Avg. Distance to Sun (AU)	0,38	0.72	1.00	1.52	5.21	9.54	19.18	30.11
Latency	3 min	6 min	8 min	**12 min**	43 min	1h20	2h40	4h10

Fig. 2.1 – *Ejemplo de almacenamiento, transporte y reenvío en una DTN espacial. Un nodo en la Tierra transmite datos a un satélite en órbita alrededor de la Luna, que los almacena hasta que esté disponible un contacto de largo alcance con Marte. Debido a a) los retrasos y b) las interrupciones, la dependencia de los mensajes de retroalimentación inmediata debe reducirse o eliminarse de las DTN. Dado que un segundo-luz se define como la distancia que recorre la radiación electromagnética en el espacio libre en un segundo, es una unidad de distancia que se corresponde trivialmente con un segundo-luz. distancia que corresponde trivialmente a una unidad de tiempo.*

2.5.2. Bundle Protocol (BP)

Especificada en la Fuerza de Tarea de investigación en Internet (IRTF, del inglés *Internet Research Task Force*) [58], y el Comité Consultivo de Sistemas de Datos Espaciales (CCSDS, del inglés *Consultative Committee for Space Data Systems*) [16], en la versión 6 del año 2007, donde se detallan los términos y formatos, se describe el servicio y la operación de este protocolo.

El diseño del *Bundle Protocol* (BP) [50] se ha implementado en base a la arquitectura DTN, en donde el *bundle* es la unidad básica de datos, un paquete que contiene datos pertinentes al protocolo de la capa de aplicación [60]. Para soportar la transferencia de datos en entornos desafiantes, como lo es el ambiente espacial, BP se basa en la modificación del modelo OSI en los puntos finales y algunos nodos intermedios de una nueva capa, que se encuentra entre la capa de aplicación y las capas inferiores llamada *Bundle Layer* [61].

El BP puede interactuar con capas inferiores diferentes a través de adaptadores CLA, que realizan la función de envío y recepción de paquetes mediante algún servicio de enlace que puede ser una red o un protocolo de Internet nativo [60].

Para comprender mejor, la figura 2.1 ilustra un ejemplo de red interplanetaria que utiliza el protocolo *Bundle* como superposición para diferentes familias de protocolos a través de CLA [38].

El nodo DTN 1, un centro de control de misión situado en la Tierra, envía datos a un rover en Marte (nodo DTN 4). Como no hay comunicación directa entre los nodos de origen y destino, los datos tienen que pasar por la estación terrestre (nodo DTN 2) hasta un satélite de retransmisión intermedio del nodo DTN 3 en órbita alrededor de la Luna. Sin embargo, la visibilidad entre los nodos 3 y 4 aún no permite establecer el enlace (por ejemplo, el rover está en el lado opuesto de Marte). Así, la rutina de enrutamiento del nodo DTN 3 decide de forma autónoma retener los datos en tránsito en su almacenamiento local hasta que el contacto con el nodo DTN 4 esté disponible, después de que el planeta gire. En tal caso, el nodo DTN 3 puede convertirse en custodio de los datos almacenados para comprometerse con el éxito de su entrega, autorizando la retirada del paquete del almacenamiento del nodo DTN 2. Asimismo, si los episodios de visibilidad son insuficientes para la transmisión de un determinado paquete utilizando un protocolo de capa de convergencia concreto, el paquete puede fragmentarse en el nodo DTN 3, en el nodo 2 o en el nodo 1. Los comandos urgentes al rover pueden utilizar clases de prioridad más altas, las alarmas telemétricas importantes pueden ser comunicadas al control de la misión a través de paquetes marcados como críticos, y el tiempo de vida (TTL,

del inglés *Time to Live*) puede establecerse en el periodo durante el cual las observaciones obtenidas de los instrumentos del rover son relevantes.

En particular, la arquitectura DTN es adecuada para actuar como superposición en la parte superior de una red heterogénea que consiste en diferentes segmentos, como las redes inalámbricas de sensores ad-hoc, Internet por cable, redes de área local inalámbricas (WLAN, del inglés *Wireless Local Área Network*) o enlaces satelitales mediante CLA. Por otro lado las características de protocolo de transporte extremo a extremo se limitan a un salto de DTN, mientras que la transferencia de datos de extremo a extremo a través de la red heterogénea es proporcionada por el *Bundle Layer* [61].

El *Bundle Protocol* aborda cinco características únicas que deben tenerse en cuenta al enrutar datos en DTN espaciales. Los elementos de datos del paquete que admiten estas funciones son:

- **Adaptadores de capa de convergencia CLA**: un CLA es el componente que envía y recibe paquetes en nombre del agente de *Bundle Protocol* que utiliza los servicios de pilas de protocolos que se adaptan bien al entorno de comunicación local. Cuando corresponda, las responsabilidades de CLA incluyen el establecimiento de sesiones, las adaptaciones de codificación, la transferencia de paquetes utilizando tamaños de unidad de transmisión máxima MTU más pequeños y otros según el protocolo nativo. Los CLA para TCP, UDP, Licklider Transmission Protocol LTP, Bluetooth y raw Ethernet permiten la integración con infraestructuras de redes espaciales e Internet existentes [62]. Para tener en cuenta los gastos generales de CLA, el consumo de volumen estimado ($B.EVC$) de un paquete B se define en [59] como el tamaño del paquete más un margen del 3 % o 100 bytes, cualquiera que sea el menor.

- **Transferencias de custodia**: para hacer frente a protocolos de capa inferior poco fiables y situaciones de congestión (agotamiento del almacenamiento), un nodo intermedio puede "tomar la custodia" de un paquete, es decir, asumir la responsabilidad de reenviarlo. Una señal de aceptación de custodia de un nodo remoto libera al custodio actual del paquete de la responsabilidad adicional de reenviarlo. El custodio puede así liberar recursos de almacenamiento y usarlos para recibir más datos.

- **Fragmentación**: dado que un paquete contiene todos los datos necesarios para una transferencia de información completa y exitosa al destino, su longitud de bytes puede ser arbitrariamente larga. Si bien la unidad de datos de protocolo UDP/IP más grande es de 64 KB, prácticamente no hay límite de tamaño para un paquete, que puede transportar MegaBytes o GigaBytes de datos de carga útil. En este contexto, la fragmentación puede garantizar un ajuste correcto a los contactos de duración determinada. La fragmentación puede ser proactiva (si ocurre antes de la transmisión del paquete) y/o reactiva (si se identifica la necesidad de fragmentación durante la transmisión del paquete).

- **Prioridades**: La especificación del protocolo de paquete reserva indicadores de clase de servicio en el bloque principal del paquete (el primer elemento del "encabezado" del paquete) que se pueden asignar a niveles de prioridad de tráfico de unicast, $(1...P)$, donde se recomienda $P = 3$ en [59]. Además, se ha propuesto un mecanismo de clase de servicio ampliado que, por separado, permite marcar un paquete como "crítico", lo que indica que las copias de los datos se reenviarán a través de tantas interfaces como sea posible para maximizar la posibilidad de que lleguen a su destino. Lo que se nombra en [63].

- **Fechas Límite**: para evitar que los datos persistan durante largos períodos de tiempo y, por lo tanto, congestionen los recursos de almacenamiento de los nodos, cada paquete se configura con una "fecha límite" o tiempo de caducidad. La "fecha límite" de un paquete se calcula como su tiempo de creación más su "vida útil" (también llamado "Tiempo de vida" TTL, pero tenga en cuenta que TTL en DTN es un límite de tiempo en lugar de un límite de salto como en la arquitectura de Internet). En particular, cuando se alcanza el tiempo de caducidad de un paquete, el agente del *Bundle Protocol* está autorizado para eliminar ese paquete de la red. La responsabilidad de establecer un plazo adecuado recae en la aplicación que origina los datos.

2.6. Enrutamiento en Redes Tolerantes a Retraso

El problema del enrutamiento en este tipo de redes es cumplir los requerimientos que los protocolos tradicionales dan por sentado.

La intermitencia de las conexiones entre nodos, como también la capacidad de cada nodo, que pueden ser parámetros no homogéneos. En otras palabras, las conexiones pueden darse durante intervalos de tiempos variables e intermitentes, y la capacidad de transmisión, recepción, procesamiento o almacenamiento de cada elemento físico de la red puede ser distinta.

Otro requerimiento tiene que ver con la capacidad del nodo. En DTN, la velocidad de transmisión, almacenamiento, transporte y retransmisión de los datos, cuando el enlace haya terminado por causa del alejamiento debido al movimiento (entre dos naves espaciales o una nave espacial y la estación terrena) el nodo receptor almacena los datos. Si la cantidad de datos excede la capacidad de almacenamiento del nodo receptor, resulta en la pérdida de paquetes. Además, cada nodo puede tener características diferentes en cuanto al poder computacional. Pueden existir nodos incapaces de ejecutar algoritmos complejos de enrutamiento, y nodos con limitada energía para funcionar.

Tradicionalmente en los los esquemas de enrutamiento, se selecciona una ruta que minimice el tiempo de entrega. Por otra parte, se debe maximizar las probabilidades de que el dato llegue a destino, por lo tanto, el objetivo es aumentar la probabilidad de entrega de un paquete, pero reducir los retardos en la entrega es igualmente importante [64].

Algunos autores han realizado diferentes clasificaciones de algoritmos de enrutamiento para DTN. En [64], el autor clasifica los algoritmos de enrutamiento en función de la cantidad de datos que utilizan éstos para calcular las rutas, es decir, información sobre los contactos, estado de la cola, y la demanda de tráfico.

Por un lado, los esquemas de reenvío son adecuados para la optimización del almacenamiento, ancho de banda y energía, ya que envían solo una copia del paquete al nodo vecino. Por otro lado, los esquemas de replicación proporcionan un mayor rendimiento en términos de probabilidad de entregar un paquete debido a que replican múltiples copias a los vecinos con la esperanza de que uno de ellos alcance al destino.

Algunos Algoritmos de Enrutamiento en DTN

En [64] el autor realiza una descripción de algunos algoritmos de enrutamiento que son empleados en DTN:

- Epidemic Routing [63]: Admite la entrega eventual de mensajes a destinos arbitrarios con suposiciones mínimas con respecto a la topología y la conectividad de la red subyacente.

Los nodos replican de forma continua y transmiten paquetes a nuevos contactos que aún no poseen una copia de éste.

Consiste en distribuir mensajes de aplicaciones a hosts, denominados "transportadores", dentro de porciones conectadas de redes ad-hoc.

Un nodo aloja mensajes en el búfer incluso si no hay ninguna ruta al destino disponible actualmente. Un índice de estos mensajes, llamado vector de resumen, es mantenido por los nodos, y cuando dos nodos se encuentran, intercambian vectores resumen. Después de este intercambio, cada nodo puede determinar si el otro nodo tiene algún mensaje que antes no se había visto en este nodo. En ese caso, el nodo solicita los mensajes del otro nodo. Esto significa que mientras haya espacio de búfer disponible, los mensajes se propagarán como una epidemia de alguna enfermedad a través de la red cuando los nodos se encuentran y se "infectan" entre sí.

De esta forma, los mensajes se distribuyen rápidamente a través de partes conectadas de la red. El enrutamiento epidémico depende entonces de que los transportadores entren en contacto con otra parte conectada de la red a través de la movilidad de los nodos. En este punto, el mensaje se propaga a una isla adicional de nodos. A través de tal transmisión transitiva de datos, los mensajes tienen una alta probabilidad de eventualmente llegar a su destino.

El objetivo general de *Epidemic Routing* es maximizar la tasa de entrega de mensajes y minimizar la latencia de entrega, al mismo tiempo que minimiza los recursos agregados del sistema consumidos en la entrega de mensajes. Esto se logra colocando un límite superior en el conteo de saltos de mensajes y el espacio de búfer por nodo (la cantidad de memoria dedicada a llevar los mensajes de otros anfitriones). Al aumentar los límites de estos parámetros, las aplicaciones pueden aumentar la probabilidad de que un mensaje se entregue con éxito a cambio de un mayor agregado consumo de recursos.

Aunque es altamente confiable, este algoritmo consume bastantes recursos, ya que no hace ningún intento por eliminar réplicas que no mejoran la probabilidad de la entrega del paquete.

En [65] se muestran los detalles de una implementación interesante en un simulador y la comparación de su rendimiento con otro protocolo.

- ProPHET [66], [67]: Esta estrategia asume que no es probable que los usuarios reales se muevan al azar, sino más bien moverse de una manera predecible basada en la repetición de patrones de comportamiento, de modo que si un nodo ha visitado un lugar varias veces antes, es probable que vuelva a visitar ese lugar.

  Para lograr esto, los autores establecen una métrica probabilística llamada predictibilidad de entrega $P(a, b) \in [0, 1]$ en cada nodo a para cada destino conocido b.

  Esto indica la probabilidad de que este nodo pueda entregar un mensaje a ese destino. Cuando dos nodos se encuentran, intercambian vectores de resumen y también un vector de predictibilidad de entrega que contiene la información de predictibilidad de entrega para destinos conocidos por los nodos.

  Esta información adicional se utiliza para actualizar el vector de predictibilidad de entrega interna. Después de eso, la información en el vector de resumen se usa para decidir qué mensajes solicitar del otro nodo en función de la estrategia de reenvío utilizada.

Lo primero que debe hacer es actualizar la métrica cada vez que se encuentra un nodo, de modo que los nodos que se encuentran a menudo tengan una alta previsibilidad de entrega.

Si un par de nodos no se encuentran por un tiempo, es menos probable que sean buenos reenviadores de mensajes entre sí, por lo que los valores de previsibilidad de la entrega deben envejecer y se reducen en el proceso.

La previsibilidad de la entrega también tiene una propiedad transitiva, que se basa en la observación de que si el nodo A se encuentra con frecuencia con el nodo B, y el nodo B con frecuencia encuentra nodo C, entonces el nodo C probablemente sea un buen nodo para reenviar mensajes destinados al nodo A.

Utilizando el historial de los encuentros y la transitividad, el conocimiento obtenido de encuentros anteriores con otros nodos se utiliza para optimizar tanto la entrega de paquetes como el rendimiento de la entrega para pronosticar contactos futuros y determinar los próximos saltos adecuados para un paquete determinado.

En [68], se muestra una mejora, empleando *Epidemic Routing*, pero seleccionando apropiadamente parámetros de entrega y saltos en los nodos.

- *Spray-and-wait* [69]:En esta estrategia se envía un número limitado de copias en la red, y luego se espera hasta que uno de estos nodos cumpla con el contacto de destino. Consta de dos fases:

  – Fase de difusión: por cada mensaje que se origina en un nodo de origen, L copias de mensajes se propagan inicialmente (reenviadas por el origen y posiblemente otros nodos que reciben una copia) a L "retransmisiones" distintas. (Los detalles sobre los diferentes métodos de rociado se darán más adelante).

  – Fase de espera: si el destino no se encuentra en la fase de rociado, cada uno de los nodos L que llevan una copia del mensaje realiza una transmisión directa (es decir, reenviará el mensaje solo a su destino).

  Inicialmente, "arranca" la difusión de copias de mensajes de una manera similar al enrutamiento epidémico. Cuando se hayan difundido suficientes ejemplares para garantizar que al menos uno de ellos encontrará el destino rápidamente (con alta probabilidad), se detiene y deja que cada nodo que lleva una copia realice una transmisión directa. En otras palabras, *Spray and Wait* podría verse como una compensación entre los esquemas de copia única y múltiple. En [69] se muestra su rendimiento.

  La definición anterior de *Spray and Wait* deja abierta la cuestión de cómo se distribuirán inicialmente las copias L. Se pueden prever varias heurísticas de "pulverización" diferentes. Por ejemplo, la forma más sencilla es hacer que el nodo de origen reenvíe todas las L copias a los primeros L nodos distintos que encuentre. Una mejor manera es la siguiente, que los autores llaman *"Binary Spray and Wait"*:

  La fuente de un mensaje comienza inicialmente con L copias; cualquier nodo A que tenga $n > 1$ *copias de mensajes (fuente o retransmisión) y encuentre otro nodo B (sin copias), entrega a B* $n/2$ *y se queda con* $n/2$*; cuando se queda con una sola copia, pasa a transmisión directa.*

  En [70] se muestra un curioso enfoque donde se reenvía copias de un mensaje de una manera *Binary Spray and Wait* modificada para que funcione incluso en una estructura de nodo no independiente e idénticamente distribuida.

- MaxProp [71]: En principio el protocolo utiliza una lista clasificada de los paquetes almacenados del par en función de un costo asignado a cada destino. El costo es una estimación de la probabilidad de entrega. Además, MaxProp utiliza los reconocimientos enviados a todos los pares para notificarles las entregas de paquetes, asigna una prioridad más alta a los nuevos paquetes y también intenta evitar la recepción del mismo paquete dos veces. Para ponderar los contactos, utiliza un método llamado promedio incremental, donde los nodos vistos con menor frecuencia reciben una menor valoración.

  En [72] se muestra un concepto nuevo que consiste en combinar el protocolo de enrutamiento MaxProp y el modelo de "transferencia por delegación" (*custody transfer*) para explotar los nodos como portadores de mensajes entre la red particionada.

  La combinación de estos dos enfoques (MaxProp y El modelo de "transferencia por delegación") combina dos tipos de técnicas de enrutamiento basadas en el grado de conocimiento que tiene el nodo sobre sus futuros contactos con otros nodos en la red.

  Cuando un nodo encuentra otro nodo con la mayor probabilidad, envía el mensaje a ese nodo y conserva el mensaje para transmitirlo a otros nodos en el futuro.

  Cuando un nodo se encuentra con otro nodo que tiene un movimiento planificado y una probabilidad baja, envía este mensaje al nodo incluso si la probabilidad es baja, luego elimina la copia del mensaje y luego libera espacio en su unidad de almacenamiento.

  Cuando un nodo se encuentra con otro nodo que tiene un movimiento planificado y una alta probabilidad de hacerlo, envía el mensaje a este nodo, luego elimina la copia del mensaje y luego libera el espacio en su unidad de almacenamiento.

- RAPID [73, 74, 75]: El enrutamiento en *RAPID* es intencional con respecto a una métrica de rendimiento determinada. *RAPID* calcula explícitamente el efecto de la replicación en la métrica de enrutamiento teniendo en cuenta las limitaciones de recursos. RAPID fue diseñado para optimizar explícitamente una métrica de enrutamiento especificada por el administrador. Enruta un paquete al replicarlo de manera oportunista hasta que una copia llega al destino. RAPID traduce la métrica de enrutamiento en utilidades por paquete que determinan en cada oportunidad de transferencia si la utilidad marginal de replicar un paquete justifica los recursos utilizados. Realiza un seguimiento flexible de los recursos de la red a través de un plano de control para asimilar una vista local del estado de la red global. Con este fin, RAPID utiliza un canal de control en banda para intercambiar información sobre el estado de la red entre nodos utilizando una fracción del ancho de banda disponible. El canal de control de RAPID se basa en conocimientos de trabajos anteriores, por ejemplo, Jain et al. [64] sugieren que los protocolos de enrutamiento DTN que usan más conocimiento de las condiciones de la red funcionan mejor, y Burgess et al. [71] muestran que la inundación de reconocimientos mejora las tasas de entrega al eliminar paquetes inútiles de la red.

- BUBBLE Rap [76], diseñado para un entorno de red tolerante a retrasos, construido a partir de dispositivos transportados por humanos en comunidades, que tiene una tasa de entrega similar, pero una utilización de recursos mucho menor que el "Flooding" (del inglés, *Inundación*), PROPHET y SimBet. BUBBLE Rap está diseñado para funcionar mejor con una estructura comunitaria jerárquica. Si un nodo tiene un mensaje destinado a otro nodo, este nodo primero "burbujearía" este mensaje en el árbol de clasificación jerárquica usando la clasificación global hasta que llegue a un nodo que tiene la misma etiqueta (comunidad) que el destino de este mensaje.

- Encounter-Based Routing (EBR) [77], protocolo de enrutamiento que logra altas tasas de entrega comparables a los protocolos basados en inundaciones, con baja sobrecarga de la red. Esta mejora en la tasa de entrega se logra prediciendo aproximadamente a partir de datos pasados. Los nodos que experimentan una gran cantidad de encuentros tienen más probabilidades de transmitir correctamente el mensaje al destino final que los nodos que solo se encuentran con otros con poca frecuencia. Dado que EBR es un protocolo de enrutamiento basado en cuotas, limita la cantidad de réplicas de cualquier mensaje en el sistema, minimizando el uso de recursos de red.

Para la aplicación de estos algoritmos es difícil predecir el momento en el que se dará un contacto debido a que están pensados para aplicaciones terrestres desatendidas, sin embargo, el monitoreo constante de los parámetros orbitales que se dan sobre las naves espaciales y satélites establece la posibilidad de estimar los contactos previamente para planear la ruta.

2.7. Red de Superposición Interplanetaria, (ION)

ION es un software diseñado para comunicaciones interplanetarias, un conjunto de comunicaciones implementaciones de protocolos de comunicación para apoyar las comunicaciones durante la misión. Está pensada originalmente para que, a través de una red interplanetaria de extremo a extremo, pueda incluir a bordo subredes (de vuelo), redes planetarias o lunares *in-situ*, enlaces de proximidad, enlaces espaciales e internet terrestres. Es una implementación de DTN para reducir costos y riesgos en las comunicaciones mediante la simplificación de la construcción y operación de las infraestructuras de redes.

El software ION contiene las siguientes distribuciones de paquetes:

- *Interplanetary Communication Infrastructure* (ICI, del inglés *infraestructura de comunicación interplanetaria*), un conjunto de bibliotecas que proporcionan soporte compatible con software de vuelo para funciones en las que los otros paquetes se basan, como la gestión de memoria dinámica, no volátil gestión de almacenamiento y comunicación entre tareas a través de memoria compartida. Las bibliotecas ici están diseñadas para realizar la migración del software IPN a múltiples sistemas operativos (Linux, VxWorks, Solaris, etc.).

- *Licklider Transmition Protocol* (LTP, del inglés *protocolo de transmisión Licklider)*, un protocolo de capa de convergencia DTN para una transmisión fiable basada en reconocimientos tolerantes a retrasos, tiempos de espera agotados, y retransmisiones. Este protocolo está definido y mejor detallado en [78].

- *Bundle Streaming Service* (BSS, del inglés *servicio de transmisión de paquete*) un paquete de servicios de transmisión (BSS) para un servicio confiable y tolerante a las interrupciones en la transmisión de datos. BSS admite aplicaciones de transmisión en tiempo real pasando las cargas útiles del paquete a la aplicación asociada para visualización de los datos más recientes mientras se almacenan todas las cargas útiles del paquete recibido en una base de datos para su reproducción dirigida por el usuario. El servicio BSS proporcionado en ION permite que un flujo de video, audio u otras unidades de datos de aplicaciones generadas continuamente, transmitidas a través de una red tolerante al retardo, se presenten a una aplicación de destino en dos modos útiles al mismo tiempo:

– En el orden en que se generaron las unidades de datos, con la menor latencia posible de entrega de un extremo a otro, pero posiblemente con algunos vacíos debido a la pérdida o corrupción transitoria de los datos.

– En el orden en que se generaron las unidades de datos, sin espacios (es decir, incluidas las unidades de datos perdidas o corruptas que se omitieron de la presentación en tiempo real pero que posteriormente se retransmitieron), pero en un modo de "reproducción" no en tiempo real.

- *Asynchronous Message Service* (AMS, del inglés *servicio de mensajes asíncronos*) una implementación del servicio de mensajes asíncronos CCSDS, en la capa de aplicación, que utiliza protocolos DTN subyacentes para la distribución de mensajes e información entre los nodos.

- *Datagram Retransmission Protocol* (DGR, del inglés *protocolo de retransmisión de datagramas*), una implementación alternativa de LTP que está diseñada para operar de manera responsable, es decir, con control de congestión incorporado, en Internet u otras redes basadas en IP. Se proporciona como un candidato "servicio de transferencia primario" en apoyo de las operaciones de AMS en un entorno similar a Internet (no tolerante a retrasos). El diseño DGR combina el concepto de LTP de transacciones de transmisión concurrentes con control de congestión y algoritmos de cálculo de intervalo de tiempo de espera adaptados de TCP.

- CCSDS *File Delivery Protocol* (CFDP, del inglés *protocolo de recuperación de archivos CCSDS*), otro servicio de capa de aplicación que no forma parte de la arquitectura DTN pero que utiliza protocolos DTN subyacentes. CFDP realiza la segmentación, transmisión, recepción, reensamblaje y entrega de archivos de una manera tolerante al retraso. La implementación de ION de CFDP se ajusta a la definición de "clase 1" del protocolo en el estándar CFDP, utilizando DTN (BP, nominalmente sobre LTP) como su capa de "transporte de datos unitarios".

- *Delay-Tolerant Payload Conditioning* (DTPC, del inglés *acondicionamiento de carga útil tolerante al retardo*), la biblioteca dtpc proporciona funciones que permiten que el software de aplicación utilice el acondicionamiento de carga útil tolerante al retardo DTPC al intercambiar información en una red tolerante al retardo. DTPC es un protocolo de servicio de aplicaciones, que se ejecuta en una capa inmediatamente por encima del protocolo *Bundle*, que ofrece soporte tolerante al retardo para varios servicios de un extremo a otro para las aplicaciones que los requieran. Estos servicios incluyen la entrega de elementos de datos de la aplicación en orden de transmisión (en lugar de recepción); detección de huecos de recepción en la secuencia de elementos de datos de aplicación transmitidos, con reconocimiento negativo de extremo a extremo de los datos faltantes; acuse de recibo positivo de un extremo a otro de los datos recibidos con éxito; retransmisión de un extremo a otro de los datos faltantes, impulsada por un acuse de recibo negativo o por la expiración del temporizador; supresión de elementos de datos de aplicaciones duplicados; agregación de elementos de datos de aplicaciones pequeños en cargas útiles de paquetes grandes, para reducir la sobrecarga del protocolo de paquetes; y elisión controlada por la aplicación de elementos de datos redundantes en cargas útiles agregadas, para mejorar la utilización del enlace.

2.8. Enrutamiento Mediante Grafos de Contacto (Contact Graph Routing)

El entorno espacial se caracteriza por tener limitado ancho de banda y limitados periodos de visibilidad radioeléctrica [40, 79, 80], por lo tanto, las redes DTN han surgido como potencial solución a la transmisión de información en dicho entorno.

Como tema de investigación, el enrutamiento en redes DTN ha sido un problema complejo causado por la conectividad intermitente y la interrupción de red frecuente, desde principios de la década de 2000. El enfoque utilizado para resolver los problemas de conectividad, fue asociar los enlaces entre satélites a un sistema de grafos cuya interacción sea determinista en un periodo de tiempo conocido.

Para generalizar la idea de enlaces intersatelitales, se debe tener en cuenta que, en el espacio, la conectividad entre satélites puede ser inconstante. Los nodos de una red de satélites o naves espaciales pueden no tener enlaces permanentes, por lo que las comunicaciones están limitadas por el tiempo intermitente de contacto radioeléctrico entre los nodos.

Uno de los aspectos más críticos dentro de la arquitectura DTN es el enrutamiento de los datos, ya que no es posible utilizar esquemas de enrutamiento tradicionales basados en conexiones estables. No es suficiente determinar el próximo salto de los datos con el siguiente nodo, desde un análisis de la topología de la red. También es necesario decidir cuando enviar los datos en la red actual; dependiendo de cuando se espera que llegue el dato a destino por medio de los enlaces y su demora.

La consideración de la dimensión temporal, es un aspecto que desafía cualquier algoritmo de enrutamiento subyacente. Afortunadamente, en las redes espaciales es posible conocer la conectividad futura de los activos [81], lo que puede expresarse en un "plan de contactos" que defina los recursos esperados que tendrá la red espacial para el transporte de datos. Estas condiciones inspiraron la creación de nuevos modelos de gráficos, abstracciones que evolucionan en el tiempo y adaptaciones de algoritmos del marco de enrutamiento de gráficos de contacto CGR.

2.8.1. Historia

Primeramente sugerido como un esquema de enrutamiento interplanetario por S. Burleigh en 2008 [2], se propusieron borradores del IETF para la CGR en 2009 [50] y 2010 [82]. En 2011 se introdujo la adaptación de Dijkstra en CGR por Segui et al. en [83]. Al beneficiarse de una función de costos relacionada con el tiempo que aumenta monótonamente, esta contribución proporcionó a la CGR un marco algorítmico sólido, en adelante más detallado.

En 2012, el Dr. Birrane presentó una visión ampliada de CGR que incluía la prevención de bucles de enrutamiento y el análisis de múltiples destinos [84].

Al mismo tiempo, el Dr.Carlo Caini et al. sugirió implementar CGR como una solución de enrutamiento para redes satelitales de órbita terrestre baja (LEO) cercanas a la Tierra [40], [85], [86].

En 2014, Bezirgiannidis et al. dio los primeros pasos en el modelado del impacto del tráfico en la estimación del tiempo de entrega de paquetes junto con técnicas de gestión de overbooking [87, 88]. Casi al mismo tiempo, Fraire et al. exploró la mitigación de la congestión mediante anotaciones de volumen adecuadas en las rutas CGR combinadas con actualizaciones de enrutamiento de origen [89, 90].

Araniti et al. resumió los avances y las experiencias experimentales con CGR hasta 2015 en [62]. En 2016, Burleigh et al. propuso una mejora oportunista de CGR para que los contactos no planificados pudieran reaccionar correctamente e incluirse en las decisiones de enrutamiento [91]. Dr. Ruhai Wang et al. luego presentó las primeras investigaciones sobre la escalabilidad de CGR [92], lo que motivó contribuciones posteriores de Madoery et al. a través de reenvío eficiente [93] y enfoques basados en regiones [94], [95]. Los estudios iniciales de confiabilidad de CGR siguieron en 2017. El Dr. Juan Fraire et al. mostró cómo se comportó CGR bajo planes de contacto inciertos [32], [96], para lo cual se introdujeron variaciones confiables de CGR basadas en modelos informáticos de última generación [97], [98], [99]. En 2019, se propuso una formulación de árbol de expansión como alternativa de CGR para calcular rutas a varios destinos [100], y se introdujo un intercambio de información de cola parcial en [101]. El intercambio de información de cola parcial se introdujo en . En ese mismo año, CCSDS publicó el estándar recomendado (libro azul) (SABR, del inglés *Schedule Aware Bundle Routing*) recomendando CGR como el procedimiento de enrutamiento para la Internet del Sistema Solar [59]. Como demuestra el desarrollo de este extenso ecosistema, CGR se ha convertido en algo más que un simple algoritmo. La CGR es un proceso integral para abordar la operación y gestión de un espacio programado DTN. Como tal, es bastante único en comparación con otros enfoques de enrutamiento, CGR es único en el sentido de que es el único enfoque (del que los autores en [38] son conscientes) que integra el conjunto de técnicas capaces de hacer frente a estos desafíos desde una perspectiva práctica.

El enrutamiento mediante grafos de contacto (CGR) surge de la necesidad de comunicación entre dos nodos de una red sin que estas estén conectadas físicamente. Para esto se aprovecha el llamado plan de contactos.

2.8.2. Contactos

La literatura [102] clasifica los contactos como:

- Oportunista: no se pueden hacer suposiciones sobre contactos futuros
- Probabilístico: los patrones de contacto se pueden inferir del historial (p. ej., redes sociales)
- Programado: los contactos se pueden predecir y documentar con precisión en un plan de contactos.

Las redes espaciales se caracterizan por tener contactos programados. Para establecer parámetros de cada contacto se aprovecha la capacidad de determinar los enlaces intersatelitales mediante algoritmos de predicción de órbitas, basados en modelos matemáticos de las variables que gobiernan el movimiento orbital (Propagadores Orbitales). Entonces un contacto posee parámetros como: tiempo de inicio, finalización, tasa de datos, e identificadores de transmisor y receptor (A y B).

Un contacto $C_{A,B}^{t_1,t_2}, R$ se define como un intervalo de tiempo $(t_1; t_2)$ durante el cual se espera que los datos sean transmitidos por DTN nodo A (el nodo de envío del contacto) a una tasa R tal que los datos serán recibidos por el nodo B (el nodo de recepción del contacto). Los valores de tiempo se pueden expresar en unidades absolutas (por ejemplo, Tiempo Universal Coordinado Gregoriano, UTCG) o en tiempo relativo con respecto a una época de referencia.

La naturaleza evolutiva y particionada de las DTN favorece la representación de la conectividad por medio de *contactos*, siendo un episodio de tiempo durante el cual un nodo es

capaz de transferir datos a otro nodo. Por lo tanto se debe notar la dimensión temporal entre los nodos.

2.8.3. Estructura de un Contacto

En [64], el autor introdujo el concepto de "contactos" basados en la característica de conectividad para discutir el problema de enrutamiento en las redes DTN.

Los nodos dentro de la red deben tener un indicador único por códigos numéricos como se sugiere en [103].

Los contactos entre nodos son definidos por parámetros 2.2 para su posterior estudio, en donde el nodo fuente es el nodo que transmitirá el paquete a un próximo nodo y el nodo destino, en el contexto de un contacto, es el nodo receptor del paquete.

Los contactos entre nodos DTN están activos solo durante intervalos de tiempo limitados conocidos como ventanas de contacto. Por lo tanto, se debe establecer el tiempo de inicio del enlace entre el nodo fuente y el nodo destino, como también el tiempo de fin del enlace.

En cada intervalo, solo una cantidad limitada de datos se puede transferir. El valor máximo es llamado volumen de contacto, el cual es el producto de la velocidad del enlace (bps) y la ventana de contactos (segundos) [86].

Si la tasa es constante, el producto $(end - start) * rate$ es suficiente para modelar la conectividad a lo largo del contacto. La tasa de datos de un contacto es la tasa media a la que se esperan datos a ser transmitida por el nodo emisor a lo largo del tiempo indicado periodo de tiempo. Y por lo tanto, tasa de datos para un contacto se puede calcular dividiendo el volumen total de datos que se pueden ser transferidos durante el contacto por la duración del contacto.

Además, la distancia aproximada, también conocida como tiempo unidireccional de luz OWLT, (del inglés *One Way Light Time*), medida en segundos luz entre los nodos A y B durante un contacto debe conocerse para poder calcular las rutas. En teoría, es posible que este valor de "rango" cambie entre el inicio y el final de un contacto, aunque para simplificar se asume que cada contacto está asociado con un solo valor de rango [59].

El último momento es el tiempo límite para enviar un paquete durante un contacto dado [50].

El tamaño de un paquete es la suma de los tamaños de todos los bloques incluyendo el bloque de datos. El consumo estimado de capacidad (ECC, del inglés *Estimated Capacity Consumption*) para un paquete puede ser calculado como la suma del tamaño del paquete y un estimado del tamaño de las cabeceras de las capas superiores.

Por otra parte, la capacidad residual de un contacto dado entre el nodo local y uno de sus vecinos, es la suma de las capacidades de ese contacto y todos los contactos programados anteriormente entre el nodo local y ese vecino, menos la suma de los ECC de todos los paquetes que están actualmente en la cola para la transmisión a ese vecino [50].

2.8.4. Plan de Contactos

El enrutamiento de contactos se da con el objetivo de transmitir paquetes de datos a pesar de la conectividad intermitente, mediante previo planeamiento de los cambios en la conectividad, donde se en lista el nodo fuente del mensaje, el nodo receptor, el inicio y final del contacto como también la capacidad del canal en una tabla similar a la tabla 2.3. Este esquema es llamado plan de contactos. Este plan de contactos es distribuido a los nodos de modo que cada nodo

Tabla 2.2 – *Parámetros de un Contacto*

Parámetros Fijos	
$C.snd$, $C.rcv$	Nodos de transmisión y recepción
$C.start$, $C.end$	Inicio y Final de tiempo de transmisión
$C.rate$	Tasa de transmisión de datos
$C.owlt$	Distancia expresada en segundos luz
$C.volume$	Volumen del contacto $((end - start) * rate)$
Parámetros Variables	
$C.MAV(p)$	Máximo volumen disponible para prioridad p
Área de trabajo en búsquedas de rutas:	
$C.arr\_time$	Tiempo de llegada de datos a destino (dst)
$C.visited$	El contacto ha sido visitado en una iteración previa?
$C.visited\_n[\,]$	Lista de nodos visitados previamente
$C.pred$	Puntero a contacto predecesor en una ruta
Área de trabajo en gestión de rutas:	
$C.suppr$	El contacto está suprimido?
$C.suppr\_nh[\,]$	Lista de próximos saltos suprimidos
Área de trabajo de reenvío:	
$C.fbtx$	Tiempo de transmisión del primer Byte para la ruta
$C.lbtx$	Tiempo de transmisión del último Byte para la ruta
$C.lbrx$	Tiempo de llegada del último Byte para la ruta
$C.EVL$	Límite efectivo de volumen para un contacto

tenga disponible la información del momento específico en el que podría comunicarse con otro nodo.

Cuando se programan cambios en la conectividad, un plan de contactos global de todos estos eventos puede ser distribuido de antemano a todos los nodos, permitiendo a cada nodo tener un conocimiento teóricamente exacto de la conectividad en la red en cualquier momento especificado.

Los mensajes del plan de contactos son de dos tipos: mensajes de contacto y mensajes de rango. Estos mensajes contienen el tiempo de inicio y finalización del contacto, el número del nodo transmisor y receptor, y la tasa de transmisión de datos programada entre los dos nodos durante este intervalo.

El plan de contactos representa cada oportunidad de contacto radioeléctrico entre dos satélites o nodos dentro de una red satelital. Puede ser diseñada previamente para obtener el mayor provecho según las necesidades y objetivos de la constelación, puesto que pueden utilizarse para tener una cobertura global o una cobertura mas limitada pero con menor tiempo de revisita.

Para una mayor cobertura global se necesitarían más contactos por órbita (en los polos) con todos los satélites de la constelación, mientras que para un menor tiempo de revisita se tendrían menores oportunidades de contacto entre los satélites puesto que tendrían la misma órbita pero un satélite seguiría al otro en esa misma órbita.

Esto debe quedar asentado en una tabla donde se pueda verificar cada contacto de cada nodo.

Para una representación coherente de los contactos y los parámetros de comunicación, los enlaces espaciales son unilaterales, por lo tanto un contacto bilateral se representa por un par de contactos unilaterales.

Tabla 2.3 – *Ejemplo de Plan de Contactos*

Ejemplo de plan de contactos					
Contacto	Desde	Hasta	Inicio	Final	Velocidad
1	A	B	1000	1150	1000
2	B	A	1000	1150	1000
3	B	D	1100	1200	1000
4	D	B	1100	1200	1000
5	A	C	1100	1200	1000
6	C	A	1100	1200	1000
7	A	B	1300	1400	1000
8	B	A	1300	1400	1000
9	B	D	1400	1500	1000
10	D	B	1400	1500	1000
11	C	D	1500	1600	1000
12	D	C	1500	1600	1000

2.8.5. Grafos

La idea de un grafo es representar relaciones entre los elementos de un conjunto de objetos o estados, según el contexto en el que se utilicen. Se representa gráficamente como un conjunto de puntos o nodos, conectados por arcos.

Matemáticamente, un *grafo* es un par $G = (V, E)$ de un conjunto, tal que, $E \subseteq [V]^2$; Por lo tanto, los elementos de E son subconjuntos de 2 elementos de V [104].

Los elementos de V son los vértices o nodos del grafo G, los elementos de E son los arcos o lineas que relacionan (o no) un nodo con otro.

Cada arista dirigida e entre dos nodos representa un encuentro "contacto" entre ellos, y se anota con una tupla (te, se), donde t es la hora del encuentro y s es el tamaño de la oportunidad de transferencia.

La carga de trabajo es un conjunto de paquetes $P = (u1, v1, s1, t1), (u2, v2, s2, t2), ...$, donde la i-ésima tupla representa el origen, el destino, el tamaño y el momento de la creación (en el origen), respectivamente, del paquete i.

Específicamente, en el caso de los grafos en redes satelitales, se considera un nodo a los satélites, estaciones terrenas o terminales de tierra que son capaces de transmitir y/o recibir datos durante un intervalo de contacto radioeléctrico.

Para resumir, lo relevante es conocer la relación que existe entre un objeto identificado (nodo) y otro, esta relación podría ser la dificultad, distancia o costo para llegar de un nodo a otro. Al relacionar las redes de comunicación con la teoría de grafos, en este contexto cada nave espacial o satélite sería un nodo dentro de la red y el costo de comunicarse con otro nodo puede llegar a ser cualquier parámetro o característica del canal de comunicación.

2.8.6. Grafo de Contactos

Desde un nodo se puede construir un grafo de contactos a partir de un plan de contactos. Según la necesidad de interconexión que exista, se puede extraer información del plan de contactos para definir una ruta que debe seguir el paquete de datos para llegar a distintos destinos, en una dimensión temporal, es decir, graficar la ruta que sigue un paquete de datos para llegar a destino en función del tiempo, teniendo en cuenta las posibles disrupciones o desconexiones que existan entre los nodos.

Este grafo construido localmente en un nodo dado, contiene por cada nodo en la red:

- Una lista de objetos llamadas *xmit* que encapsulan el tiempo de inicio y finalización del contacto, el número del nodo de transmisión y la velocidad de transmisión, derivados todos de los mensajes de contacto. Esta lista es ordenada por el tiempo de finalización.
- Una lista de objetos llamadas origen que encapsulan el número del nodo de transmisión y la distancia actual de ese nodo desde el receptor.

Es importante recalcar que esta información debe ser actualizada a medida que pasa el tiempo, es decir, los objetos *xmit* y *origin* cuyos tiempos de finalización se han alcanzado deben ser eliminados del grafo.

En general, para encontrar todas las rutas posibles desde un origen hasta un destino, CGR utiliza una expresión de grafo de contacto del plan de contacto. Un grafo de contacto es un grafo acíclico dirigido conceptual cuyos vértices corresponden a contactos, mientras que los arcos representan episodios de retención de datos (es decir, almacenamiento) en un nodo [105].

2.8.7. Enrutamiento (Procedimiento)

El enrutamiento en redes espaciales requiere una determinación precisa de cuándo y a qué nodo vecino se debe reenviar un paquete determinado. Debido a que los episodios de conectividad y sus respectivos retrasos de propagación pueden predecirse y estar disponibles con anticipación, el enrutamiento en las redes espaciales puede ser preciso y eficiente. El determinismo y la disponibilidad de información en las redes espaciales contrasta con las DTN oportunistas y probabilísticas, donde el enrutamiento es menos seguro pero puede basarse en métodos de inferencia de probabilidad o inundación *flooding* más simples [66, 71, 69, 106]. En cambio, la información de conectividad de la red espacial codificada en el plan de contacto debe procesarse de manera eficiente para determinar las rutas candidatas precisas. Los modelos teóricos (p. ej., modelos de programación lineal) pueden garantizar resultados óptimos a expensas del procesamiento [64], un hecho que limita cualquier valor práctico, especialmente cuando se consideran computadoras de a bordo con recursos limitados. Otro límite de los modelos de programación lineal es el requisito de información precisa sobre el estado de los nodos remotos, que normalmente no está disponible en una red tolerante a retrasos.

Para lograr la transmisión exitosa de los datos el enrutamiento en redes DTN programadas sigue una serie de estados: Planificación, Determinación y Reenvío.

- **Planificación**

  En la etapa de planificación, una entidad centralizada (por ejemplo, el control de la misión) calcula los planes de contacto en función de la estimación de las futuras ventanas de comunicaciones. Esta tarea emplea algoritmos de propagación orbitales que predicen la disposición física y la orientación de los nodos, así como los modelos y configuraciones del sistema de comunicación de las misiones (antena, modulación, potencia de transmisión, etc.). El plan de contacto resultante comprende el momento en el cual la conectividad de la red puede ocurrir. Luego, ese plan se puede procesar posteriormente para acomodar los planes operativos (episodios anticipados de desconexión debido a la administración de energía, apuntamiento de instrumentos fijos en el cuerpo, etc.) y para mejorar la equidad [107], adaptarse al enrutamiento obligatorio [108] acomodar flujos de tráfico conocidos [109], [110], mitigar la congestión [111], reducir el consumo de energía [112], o adaptarse a misiones específicas [113].

- **Determinación**

El plan de contacto sirve como entrada para la rutina de enrutamiento con CGR en su núcleo. Los enfoques algorítmicos para CGR se aprovechan para calcular las rutas a los destinos en la red. Las rutas resultantes no solo identifican a qué nodo de siguiente salto reenviar el paquete, sino que también indican el mejor tiempo de entrega (BDT), el límite de volumen de la ruta ($volumen$) y el intervalo o ventana de oportunidad, en el que esta ruta es válida para la transmisión (tx_win). Entre otras, estas métricas son calculadas por CGR y almacenadas en tablas de rutas. En el caso de que las rutas se calculen en un nodo centralizado, las tablas resultantes deberán ser distribuidas a los nodos de la DTN en el momento oportuno.

- **Transmisión**

  Finalmente, el proceso de envío o transmisión es responsable de seleccionar la mejor ruta, entre muchas en la tabla de rutas. Esta selección tiene en cuenta las condiciones locales que solo están disponibles en ese momento de reenvío, como la hora local, el tamaño y la prioridad de los datos que se reenviarán ($B.EVC$, $B.priority$) y las condiciones actuales de acumulación de cola. Por lo tanto, se espera que la mejor ruta proporcione los recursos adecuados para la entrega exitosa del paquete. Según la ruta seleccionada, el paquete se coloca en la cola de salida hacia el nodo que se identifica como el siguiente salto en la ruta. Una vez en la cola, el paquete puede transmitirse inmediatamente o almacenarse hasta que se produzca el siguiente contacto.

Una ruta bien formada para un paquete dado, se define como una secuencia de contactos, tal que, el primer contacto es desde el nodo fuente del paquete a algún otro nodo, cada contacto subsiguiente, en la secuencia, es desde el nodo receptor del contacto anterior a algún otro nodo. El último contacto de la secuencia es desde algún nodo hasta el nodo de destino final del paquete, y que la ruta no contenga bucles, es decir, no hay dos contactos en la secuencia que impliquen transmisión desde el mismo nodo y tampoco dos contactos en la secuencia que impliquen transmisión al mismo nodo [50].

El método para asignar una secuencia de nodos para la transmisión de los datos (enrutamiento), en CGR, consiste en calcular localmente las rutas que el paquete debe tomar, utilizando la información del plan de contactos almacenada en cada nodo. Esto es posible teniendo información actualizada periódicamente.

Para cada paquete de datos, el algoritmo CGR calcula rutas completas, con parámetros específicos 2.4, hacia el destino y selecciona el mejor camino teniendo en cuenta las restricciones que existan, como algún nodo no cooperativo o no disponible para recibir el paquete. Como la selección del mejor nodo próximo al que se enviará el paquete debe ser actual, por los cambios que pudo haber, se efectúa nuevamente el cálculo de la ruta y la selección de un nuevo mejor nodo próximo.

A su vez, el nodo próximo seleccionado recalcula nuevamente la ruta cuando el paquete se recibe [86]. Cabe mencionar que este algoritmo no almacena las rutas ya calculadas para cálculos posteriores.

Entonces cada nodo es capaz de resolver, es decir, encontrar la siguiente ruta para transmitir el paquete de dato en una red cuya topología varía constantemente en el tiempo. En vez de tener una ruta definida, en CGR, el salto del paquete de datos al siguiente nodo es resuelta y definida en cada nodo, por lo que brinda la posibilidad de adaptarse a los cambios en la red y la demanda de los usuarios [32].

Tabla 2.4 – *Parámetros de Rutas*

Parámetros Fijos	
$R.hops[\,]$	Lista de contactos en la ruta
$R.to\_node$	Nodo final en la ruta ($hops[-1].dst$)
$R.next\_node$	Primer vecino en la ruta ($hops[1].dst$)
$R.tx\_win(s,e)$	Intervalo de tiempo (s,e) donde la ruta es válida
$R.BDT$	El mejor al cual el dato puede llegar a destino dst
$R.volume$	Cantidad máxima de volumen de datos que la ruta puede transmitir
Parámetros variables	
Área de trabajo de reenvío:	
$R.ETO$	Oportunidad temprana de transmisión
$R.PAT$	Tiempo de llegada calculado
$R.EVL$	Limite de volumen efectivo de la ruta

2.9. Algoritmo CGR

El algoritmo CGR permite el cálculo de rutas en redes DTN entre un nodo fuente y destino, donde se asume que la conectividad entre nodos es programada e intermitente [82]. Para esto, CGR sigue una secuencia de pasos detallada a continuación:

I Se crea un plan y un grafo de contactos en función de los mensajes de contacto y rango.

II El enrutamiento CGR ejecuta recursivamente el algoritmo CGR-CRP (del inglés, *Contact Review Procedure*), el cual genera un listado de nodos próximos de un salto capaces de entregar el paquete a destino mediante varios saltos de nodos en la ruta, a través del envío a nodos sucesivos de una ruta, hasta llegar al destino final.

III El algoritmo CGR-FBP (del inglés, *Forward Bundle Procedure*) selecciona un nodo del listado de nodos próximos en función de las métricas para el envío de los paquetes.

La estructura de datos utilizada para este algoritmo se muestra en 2.2. La principal ventaja de las estructuras de datos en CGR es que se pueden utilizar como entrada para distintos algoritmos tradicionales de selección de la mejor ruta. En este trabajo, el algoritmo de mejor ruta (menor costo) utilizado es el Dijkstra, y se puede adaptar para encontrar una ruta desde un nodo de origen a un nodo de destino con el mejor tiempo de entrega en un gráfico de contacto[83]. El resultado puede utilizarse para determinar el siguiente salto o servir como ruta en una lista de rutas para luego utilizar otros criterios de decisión.

2.9.1. Algoritmo de Dijkstra

El algoritmo de Dijkstra (Algoritmo 1), [114] es utilizado comúnmente en grafos para determinar el menor costo (*Shortest Path*) cuando los costos son invariantes en el tiempo, además, se obtienen como resultado no solo la ruta de menor costo, sino que todas las rutas que llegan a destino.

En este sentido, el tiempo de llegada del dato a cada nodo debe ser conocido y esos tiempos conocidos se deben usar para determinar subsecuentemente cada salto hasta llegar a destino.

Considerando un grafo ponderado, del cual se tiene un nodo de inicio y un nodo destino, cuyo costo para llegar a destino sea el tiempo, el algoritmo se encarga de "visitar" cada nodo y cargar en memoria cada costo de un nodo a otro sin repeticiones en distintas iteraciones y así conocer las rutas que lleguen a destino que posteriormente se almacenan en una lista.

Algorithm 1: Búsqueda de Dijkstra en Grafos de Contacto

Data: contacto raíz C_{raiz}, destino D, plan de contacto PC (Area de trabajo reiniciada)
Result: Rutas R_S^D desde fuente S a destino D

```
1  R_S^D ← {}
2  C_fin ← {}                                              // contacto final
3  BDT = ∞                                                 // tiempo de llegada final
4  C_actual = C_raiz                                       // Contacto actual es el contacto raíz
   /* Ciclo de exploración del plan de contactos */
5  while 1 do
       /* Estudio del contacto */
6      C_fin, MTE ←CRP (CP, C_actual, C_fin, MTE)
       /* Procedimiento de selección del contacto */
7      C_siguiente ←CSP (CP, C_actual, MTE)
8      if C_siguiente ≠ {} then
9          C_actual ← C_siguiente
10     else
11         break                                           // Estudio y selección completados
   /* Ciclo de construcción de las rutas */
12 if C_fin ≠ {} then
13     C = C_fin
14     while C ≠ C_{S,S}^{0,∞} do
15         R_S^D.saltos ← {C}
16         C = C.predecesor                                // ruta del contacto previo
17     Calcular(R_S^D.tx_win, R_S^D.volume)
```

2.9.2. Algoritmo de Procedimiento de Revisión de Contactos CRP

El algoritmo CRP en 2 forma parte del algoritmo de Dijkstra, y analiza el plan de contactos en busca de nodos vecinos para un siguiente salto del paquete, dentro de una serie de saltos hasta llegar a destino, actualizando el área de trabajo de los contactos sucesores del contacto actual. Para encontrar estos nodos, se descartan los nodos en los que:

- El tiempo de contacto sea menor al tiempo de arribo.
- El contacto ya ha sido visitado (o analizado) anteriormente.
- El contacto continúa con otro que ya ha sido visitado.
- El contacto continúa con otro que ha sido suprimido por el administrador de la red.

Luego el algoritmo determina el mejor tiempo de arribo para el paquete y continua en un ciclo de decisión para mejorar el tiempo de arribo en el área de trabajo del contacto actual. En este punto también se marca si se llegó al nodo destino.

2.9.3. Algoritmo de Procedimiento de Selección de Contactos CSP

El algoritmo de selección CSP en 3 también forma parte del algoritmo de Dijkstra. Una vez que el plan de contactos haya sido revisado y el área de trabajo haya sido actualizado por CRP, el procedimiento en CSP vuelve a recorrer el plan de contactos para encontrar cual de los contactos posee el mejor tiempo de arribo a destino hasta que no queden contactos posteriores, evitando los contactos suprimidos, ya visitados y los que tienen un tiempo de entrega mayor al actual. Es en este punto donde debe mantenerse la suposición de Dijkstra de una métrica

Algorithm 2: Contact Review Procedure (CRP)

Data: PC, C_{actual}, C_{fin}, MTE
/* Plan de contactos, Contacto actual, Contacto final, Mejor tiempo de entrega */
Result: PC revisado, C_{fin}, MTE
1 **for** *contacto* $C \in PC \mid C.fuente = C_{actual}.destino$ **do**
  /* ignorar condiciones del test */
2 **if** $C.fin \leq C_{actual}.tiempo\_llegada$ **then**
3 continuar C // (i) ignorar contactos programados
4 **if** $C.visitado$ **then**
5 continuar C // (ii) ignorar contactos visitados
6 **if** $C.a \in C_{actual}.visitado\_n$ **then**
7 continuar C // (iii) ignorar nodos visitados
8 **if** $C.suprimido$ **or** $C \in C_{actual}.suprimido\_nh$ **then**
9 continuar C // (v) ignorar contactos suprimidos
  /* Calcular tiempo de llegada */
10 **if** $C.inicio < C_{actual}.tiempo\_llegada$ **then**
11 $tiempo\_llegada = C_{actual}.tiempo\_llegada$
12 **else**
13 $tiempo\_llegada = C.inicio$
14 $tiempo\_llegada += C.owlt + owlt_{mgn}$
  /* owlt ->segundos luz */
  /* Actualizar tiempo de llegada si mejora */
15 **if** $tiempo\_llegada < C.tiempo\_llegada$ **then**
16 $C.tiempo\_llegada = tiempo\_llegada$
17 $C.pred = C_{actual}$
18 $C.visitado\_n = C_{actual}.visitado\_n + C.dst$
  /* Mark if destination reached */
19 **if** $C.destino = D$ **and** $C.tiempo\_llegada < MTE$ **then**
20 $MTE = C.tiempo\_llegada$
21 $C_{fin} = C$
22 $C_{actual}.visited = verdadero$ // contact review completado

de costos que aumenta monótonamente. De hecho, dado que el tiempo no puede retroceder, no tiene sentido continuar con la exploración a través de tal contacto.

En [38] se tiene una descripción detallada de los algoritmos mencionados.

Algorithm 3: Contact Selection Procedure (CSP)

Data: PC, BDT
Result: $C_{siguiente}$

```
1  C_siguiente ← {}
2  t_llegada_mas_temprana = ∞                          // menor tiempo de entrega
3  for contacto C ∈ PC do
4  |  if C.suprimidos or C.visitado then
5  |  |  continuar C                                   // ignorar suprimidos o visitados
6  |  if C.tiempo_llegada > BDT then
7  |  |  continuar C                                   // ignorar el peor tiempo de llegada
8  |  if C.tiempo_llegada < t_llegada_mas_temprana then
9  |  |  t_llegada_mas_temprana = C.tiempo_llegada
10 |  |  C_siguiente =← C
```

CAPÍTULO 3

Implementación

En este capítulo se introduce detalladamente la implementación de la subrutina CGR, de una distribución de software de mayores dimensiones como lo es ION, que a su vez es una puesta en marcha de la arquitectura DTN, preparada para correr en naves espaciales para comunicaciones interplanetarias.

El software utilizado para el desarrollo del código fue el entorno de desarrollo integrado (IDE) de Eclipse (Oracle), en lenguaje C, de manera reducida y condensada. El resultado es un código fuente que cumple los requerimientos del algoritmo y es capaz de correr en la OBC NanoMind A712c que se puede ver en la figura 3.2.

3.1. Características de la NanoMind A712c

El ordenador de a bordo NanoMind está diseñado como un sistema eficiente para aplicaciones espaciales con recursos limitados, como para misiones Cubesat o nanosatélites. Trae incorporado un sistema de sensores específicos para la aplicación, un magnetómetro de 3 ejes para detectar el campo magnético de la Tierra y controladores de bobina que funcionan como "*magnetorquers*" que se pueden utilizar para implementar el control de actitud basado en la detección y activación de campo magnético. Su principal interfaz con otros subsistemas es un bus CAN y un bus I2C.

En la hoja de datos de la *NanoMind* OBC [115] se pueden extraer las características más relevantes para el tema de investigación:

- CPU ARM7 RISC de 32 bits de alto rendimiento [116]
- Compatible con FreeRTOS y *eCos*, sistemas operativos en tiempo real
- Velocidad de reloj: 8-40 MHz
- RAM estática de 2 MB
- Almacenamiento de datos de 4 MB (memoria flash)
- Almacenamiento de código de 4 MB (memoria flash)
- Compatibilidad con tarjeta MicroSD de 2GB
- Conector de bus CubeSatKit de 104 pines
- RTC: reloj en tiempo real con energía de respaldo

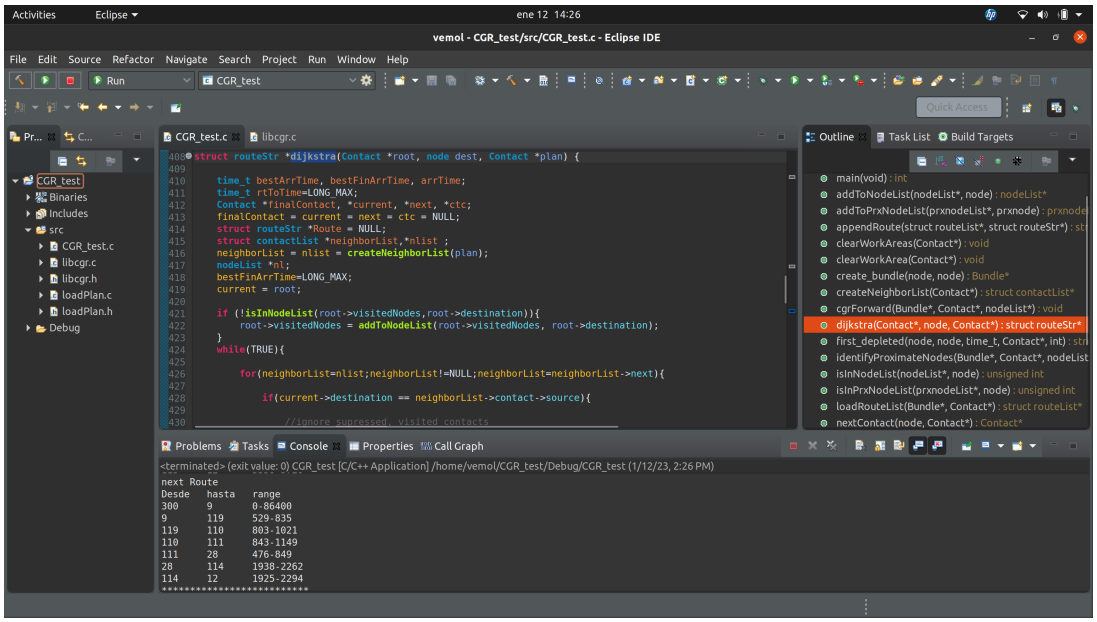


Fig. 3.1 – *Entorno de desarrollo Eclipse, utilizado para la implementación y depuración del código CGR, como también para la carga de datos en la OBC NanoMind.*

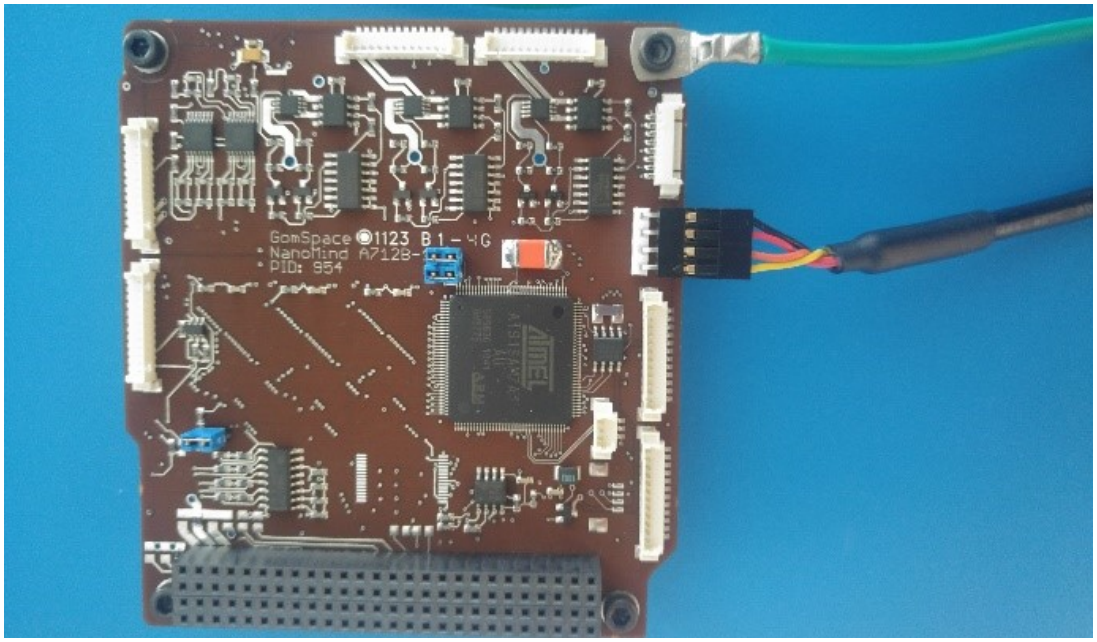


Fig. 3.2 – *NanoMind A712c*

3.1.1. Procesador

La computadora se basa en el procesador integrado ARM7TDMI [116]. Este procesador es de Arquitectura RISC de 32-bits de alto rendimiento con un conjunto de instrucciones de 16-bits de alta densidad y muy bajo consumo. La arquitectura ARM se basa en los principios de la computadora con conjunto de instrucciones reducido (RISC).

Se utiliza una canalización (pipeline) de tres etapas, por lo que las instrucciones se ejecutan en tres etapas: extracción, decodificación y ejecución. Durante el funcionamiento normal, mientras se ejecuta una instrucción, se decodifica su sucesora y se recupera una tercera instrucción de la memoria.

La versión TDMI está diseñada para admitir conjuntos de instrucciones ARM (32-bits) y *Thumb* (16-bits), un mejor desempeño en la multiplicación de registros; está preparada para los procesos de depuración, incluyendo herramientas de *hardware* para pruebas, suspensión del procesador. Estas siglas representan:

T: Admite conjuntos de instrucciones ARM (32 bits) y Thumb (16 bits) El conjunto de instrucciones ARM original consta de códigos de operación (opcodes) de 32 bits, por lo que el patrón binario para cada operación posible tiene una longitud de cuatro bytes.

Para mejorar la densidad del código, se desarrolló un conjunto de instrucciones nuevo y más pequeño llamado "Thumb", que implementa las partes más utilizadas del conjunto de instrucciones ARM pero las codifica en un patrón de 16 bits o 2 bytes (u ocasionalmente, un par de dichos códigos de operación).

Los procesadores ARM más nuevos admiten versiones mejoradas y extendidas de uno o ambos conjuntos de instrucciones en arquitecturas ARMv5, ARMv6 y ARMv7, o pueden admitir un nuevo conjunto de instrucciones de 32 bits para una arquitectura de ruta de datos de 64 bits en ARMv8, ya sea en lugar de o junto con Conjuntos de instrucciones compatibles con ARM y Thumb laterales.

D: Contiene extensiones de depuración Las extensiones de depuración proporcionan el mecanismo mediante el cual se puede suspender el funcionamiento normal del procesador para la depuración, incluidos los puertos de señal de entrada para desencadenar este comportamiento; por ejemplo, una señal para permitir que se indique un punto de interrupción y una señal para permitir que se indique una solicitud de depuración externa.

M: Bloque multiplicador 32x8 mejorado Los procesadores ARM anteriores (anteriores a ARM7TDMI) usaban un bloque multiplicador más pequeño y simple que requería más ciclos de reloj para completar una multiplicación. La introducción de este multiplicador 32x8 más complejo redujo la cantidad de ciclos necesarios para una multiplicación de dos registros (32 bits * 32 bits) a unos pocos ciclos (dependiendo de los datos). Los procesadores ARM modernos generalmente son capaces de calcular al menos un producto de 32 bits en un solo ciclo, aunque algunos de los procesadores Cortex-M más pequeños brindan una opción de implementación más rápida (ciclo único) o más pequeña (ciclo 32) 32- Bloque multiplicador de bits.

I: Macro celda ICE integrada La macro celda "*EmbeddedICE*" consta de una lógica "*on-chip*" para admitir operaciones de depuración. En ARM7TDMI-S, esto incluye dos comparadores de punto de interrupción de instrucción y punto de observación de datos, un registro de estado de cancelación y un canal de comunicaciones de depuración para pasar datos entre el objetivo y el "*host*". "*EmbeddedICE*" interactúa con las extensiones de depuración, por ejemplo, para indicar que el procesador se detiene cuando se alcanza un punto de interrupción.

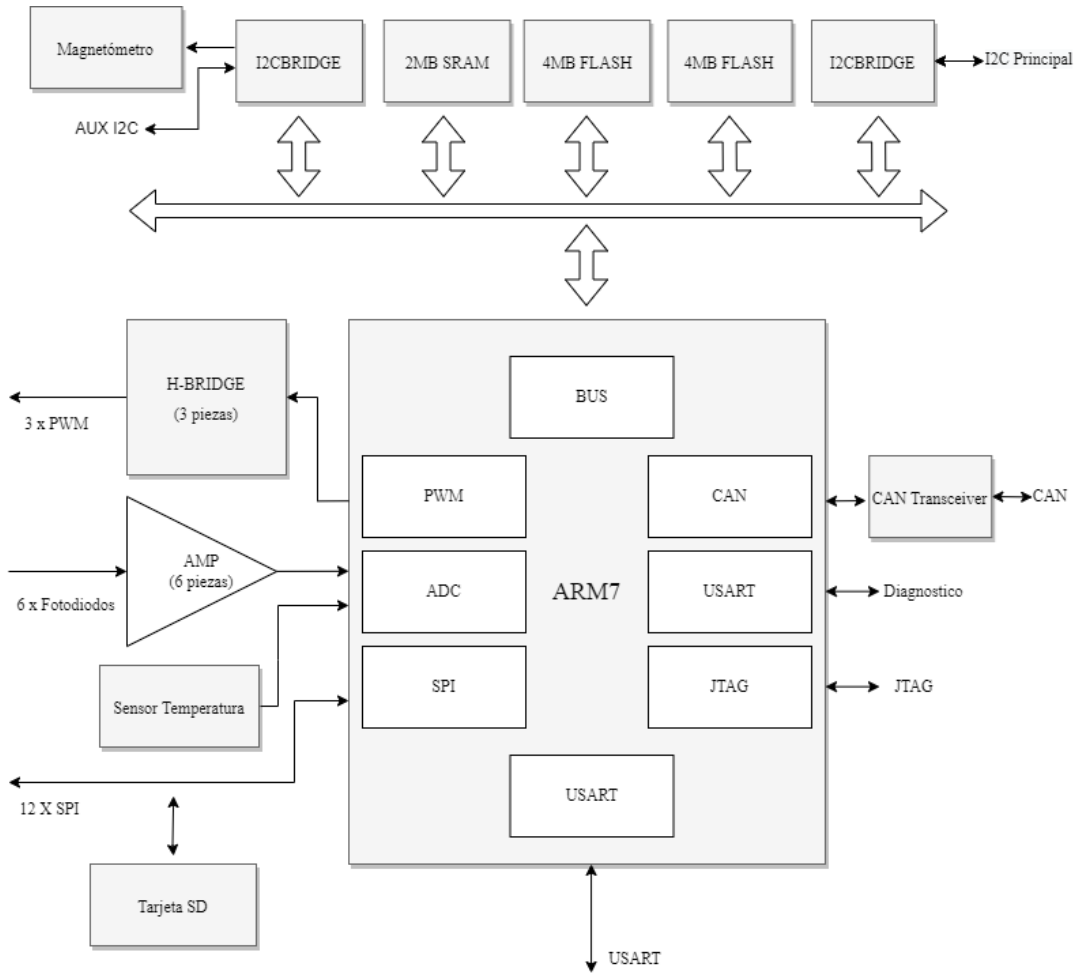


Fig. 3.3 – *Diagrama de bloque de la NanoMind A712c*

3.1.2. Memoria

En cuanto al acceso a memoria, el núcleo ARM7TDMI tiene una arquitectura Von Neumann, con un solo bus de datos de 32 bits que transporta instrucciones y datos. Solo las instrucciones de carga, almacenamiento e intercambio pueden acceder a los datos de la memoria.

La interfaz de memoria del procesador se ha diseñado para permitir que se alcance el potencial de rendimiento, al mismo tiempo que se minimiza el uso de la memoria. Las señales de control de velocidad crítica se canalizan para permitir que las funciones de control del sistema se implementen en la lógica estándar de baja potencia. Estas señales de control facilitan la explotación de los modos de acceso de ráfaga rápida compatibles con muchas tecnologías de memoria en chip y fuera de chip.

El núcleo ARM7TDMI tiene cuatro tipos básicos de ciclo de memoria:

- Ciclo inactivo
- Ciclo no secuencial
- Ciclo secuencial
- Ciclo de transferencia de registros del coprocesador

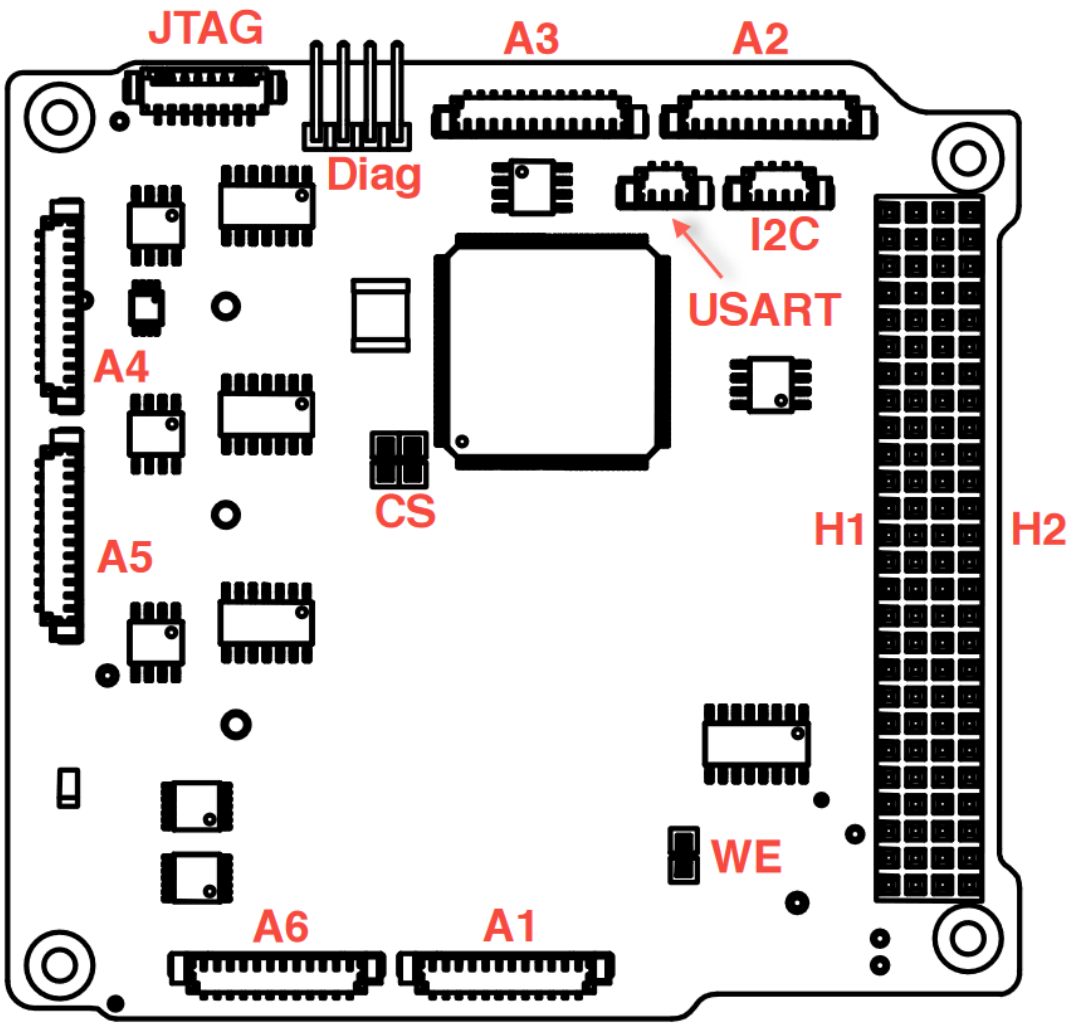


Fig. 3.4 – *Conectores en la placa electrónica NanoMind A712c*

3.1.3. Conectores

En [115] se muestra un diagrama de conexiones de la NanoMind:

- H1+H2: Conector de 104 pines CubeSatKit (SAMTEC ESQ-126-49-G-D o compatible)
- A1-A6: Señales para fotodiodos externos, giroscopios, sensor de temperatura y magnetorquers. La interfaz es totalmente compatible con los productos GomSpace NanoPower Solar Panel
- WE: Jumper para proteger la memoria flash de almacenamiento de código
- Diag: Módulo de entrada de diagnóstico
- USART: Interfaz USART adicional a la de diagnóstico.
- I2C: Acceso a la interfaz I2C utilizada por el magnetómetro
- JTAG: Interfaz de programación y depuración

3.1.4. Protocolos de Comunicación

Cubesat Space Protocol

Cubesat Space Protocol (CSP) es un pequeño protocolo de entrega de capa de red diseñado para *Cubesats*. CSP permite que los sistemas integrados distribuidos tengan una topología de red orientada a servicios. La estratificación de CSP corresponde a las mismas capas que el modelo TCP/IP. La implementación consta de un protocolo de transporte orientado a la conexión (capa 4), un núcleo de enrutador (capa 3) y varias interfaces de red (capa 1-2). Una topología orientada a servicios facilita el diseño de subsistemas satelitales, ya que el propio bus de comunicación es la interfaz con otros subsistemas. Esto significa que cada desarrollador de subsistemas solo necesita pensar en definir un contrato de servicio y un conjunto de números de puerto a los que responderá su sistema. Además, se reducen las interdependencias de los subsistemas y se agrega fácilmente la redundancia agregando múltiples nodos similares al bus de comunicación. [**?**]

USART

Los sistemas integrados, los microcontroladores y las computadoras utilizan principalmente UART (asíncrono) como una forma de protocolo de comunicación de *hardware* de dispositivo a dispositivo. Según la configuración usada, UART puede implementarse en diversas capas físicas para comunicación serial. Para una comunicación bidireccional, se utilizan dos hilos para que la transferencia de datos serie se realice correctamente. Dependiendo de la aplicación y los requisitos del sistema, las comunicaciones serie necesitan menos circuitos y cables, lo que reduce el coste de implementación.

A pesar de ser un método ampliamente utilizado de protocolo de comunicación de *hardware*, no está completamente optimizado todo el tiempo. La implementación adecuada del protocolo comúnmente se ignora cuando se usa el módulo UART dentro del microcontrolador.

Por definición, UART es un protocolo de comunicación de "*hardware*" serial asíncrona con velocidad configurable. Asíncrono significa que no hay señal de reloj para sincronizar los bits de salida del dispositivo de transmisión que van al extremo receptor.

Un USART, por otro lado, se puede configurar para ejecutarse en modo síncrono. En este modo, el periférico de envío generará un reloj que el periférico de recepción puede recuperar del flujo de datos sin conocer la velocidad en baudios de antemano. Alternativamente, el enlace usará una línea completamente separada para transportar la señal del reloj. El uso del reloj externo permite que la tasa de datos del USART sea muy superior a la de un UART estándar, llegando hasta tasas de 4 Mbps.

La interfaz de diagnóstico (DIAG) y la principal (USART Picoblade) utilizan el puerto USART en modo UART con una velocidad de reloj de 500k baudios, 8 bits de tamaño de palabra, sin bits de paridad y un bit de parada [115].

I2C - SPI

El protocolo SPI, del inglés (*Serial Peripheral Interface*) y I2C (del inglés *Inter-Integrated Circuit*), son protocolos muy utilizados para la comunicación entre microchips de propósito general con los microcontroladores.

Un beneficio único de SPI es el hecho de que los datos se pueden transferir sin interrupción. Se puede enviar o recibir cualquier número de bits en un flujo continuo. Con I2C y UART, los datos se envían en paquetes, limitados a un número específico de bits. Las condiciones de inicio y fin definen el comienzo y el final de cada paquete, por lo que los datos se interrumpen durante la transmisión.

Los dispositivos que se comunican a través de SPI o I2C están en una relación maestro-esclavo. El maestro es el dispositivo de control (generalmente el microcontrolador), mientras que el esclavo (generalmente un sensor, una pantalla o un chip de memoria) recibe instrucciones del maestro. La configuración más simple es un sistema de un solo maestro y un solo esclavo, pero un maestro puede controlar más de un esclavo.

La señal de reloj sincroniza la salida de bits de datos del maestro con el muestreo de bits por parte del esclavo. Se transfiere un bit de datos en cada ciclo de reloj, por lo que la velocidad de transferencia de datos está determinada por la frecuencia de la señal de reloj. La comunicación SPI siempre la inicia el maestro, ya que el maestro configura y genera la señal del reloj.

El A712D tiene dos selecciones de chip SPI en cada panel lateral. Al usar los paneles laterales *GomSpace NanoPower* Solar P100U, la selección de chip SPI en el pin dedicado (5) se usa para un *Gyro* y la selección de chip en el pin 12 para un sensor de temperatura LM70. Además, se utiliza un canal SPI para la interfaz de la tarjeta SD.

Con I2C se utilizan dos conductores, a diferencia de los 3 que utiliza SPI. I2C no tiene líneas de selección de esclavos como SPI, por lo que necesita otra forma de hacerle saber al esclavo que se le envían datos, y no a otro esclavo. Lo hace dirigiéndose al esclavo de interés enviando primeramente la dirección del esclavo con que se desea intercambiar datos. El byte de dirección es siempre el primer byte después del bit de inicio en un mensaje nuevo.

A su vez,los datos se transfieren en mensajes. Los mensajes se dividen en marcos de datos. Cada mensaje tiene una trama de dirección que contiene la dirección binaria del esclavo y una o más tramas de datos que contienen los datos que se transmiten. El mensaje también incluye condiciones de inicio y parada, bits de lectura/escritura y bits de confirmación ACK/NACK entre cada trama de datos.

La interface I2C trabaja a 400 kbps y se utiliza para la comunicación con el magnetómetro.

JTAG

Los avances en el diseño de chips de silicio, como el aumento de la densidad del dispositivo y, más recientemente, el empaque BGA han reducido la eficacia de los métodos de prueba tradicionales, donde cada componente se prueba individualmente. Esto es difícil de lograr en chips de alta densidad como los sistemas embebidos o SoC, del inglés *System on a Chip*.

Para superar estos problemas, algunos de los principales fabricantes de chips de silicio del mundo se unieron para formar el Grupo de Acción de Prueba Conjunta (del inglés *Joint Test Action Group*). Los hallazgos y recomendaciones de este grupo se usaron como base para el estándar 1149.1 del Instituto de Ingenieros Eléctricos y Electrónicos (IEEE): puerto de acceso de prueba estándar y arquitectura de escaneo de límites. Este estándar ha conservado su vínculo con el grupo y se conoce comúnmente por el acrónimo JTAG.

La principal ventaja que ofrece la utilización de la tecnología de exploración de límites es la capacidad de establecer y leer los valores en los pines sin acceso físico directo.

Todas las señales entre la lógica del núcleo del dispositivo y los pines son interceptadas por una ruta de escaneo en serie conocida como Registro de escaneo de límites (BSR) que consta de una serie de çeldas"de escaneo de límites. En funcionamiento normal, estas celdas de exploración de límites son invisibles. Sin embargo, en el modo de prueba, las celdas se pueden usar para configurar y/o leer valores de los pines del dispositivo (o en el modo 'interno' de los valores de la lógica central). No todas las celdas de exploración de límites son iguales: hay 10 tipos de celda en los estándares 1149.1, aunque los fabricantes pueden definir tipos de celdas no estándar para que coincidan con la funcionalidad de *hardware* real de su dispositivo si así lo desean.

La carga del programa y el acceso a los registros se realiza mediante este protocolo y se detallará más adelante.

3.1.5. Herencia de Vuelo

Esta OBC se ha usado desde el 2012 en misiones como:

- Estado operacional confirmado en la nave de ESA Vega maiden flight 13/2-2012. [115]
- SSAU *Nanosatellite Project for the Navigation and Control Technologies Demonstration* [117]
- *The STRaND-1 Nanosatellite* [118]
- *GOMX-1 Flight Experience and Air Traffic Monitoring Results* [119]
- Dellingr: NASA *Goddard Space Flight Center's First 6U Spacecraft* [120]

3.1.6. Sistema Operativo

Se puede escribir una excelente rutina de ejecución para un sistema embebido, pero en proyectos más complejos, donde exista una lista de tareas independientes a realizar, con mayor cantidad de dispositivos conectados a un mismo microprocesador, necesitando acceso a memoria y periféricos, es beneficioso utilizar un planificador de tareas (*scheduler*) para gestionar los tiempos y ejecución de tareas.

Existen varias ventajas en el desarrollo de código para diferentes subsistemas de un satélite en un sistema operativo en tiempo real:

- Permite las multitareas, manteniendo una sincronización abstracta.
- Permite la planificación de tareas mediante prioridades
- Permite la sincronización de acceso a recursos
- Existe comunicación entre tareas
- Predecible en el tiempo
- Permite la portabilidad del código de aplicaciones a otra CPU. Un sistema operativo gestiona los recursos de *hardware*, facilitando las tareas de desarrollo.
- Permite el ahorro del tiempo de desarrollo, enfocado directamente a las aplicaciones.

El fabricante provee la OBC *NanoMind* preparada para operar con el sistema operativo en tiempo real FreeRTOS como también eCos. También se incluyen *Frameworks* de software, que son herramientas de código para permitir una rápida puesta en marcha de este "*hardware*" adicional a bordo. Dependiendo del *hardware* adicional adquirido, como el subsistema de potencia EPS, el subsistema de comunicaciones COM, y el subsistema de control y determinación de actitud ADCS.

FreeRTOS es de código abierto, principalmente escrito en lenguaje C, con algunas partes en lenguaje ensamblador. Con la habilidad de crear, pausar o suspender tareas programadas para ejecutar tareas de mayor prioridad, (lo que se llama *preemptive real-time operating system*) para volver a ejecutarlas cuando haya finalizado la tarea de mayor prioridad.

Está disponible para la descarga como una librería de tipos y funciones para desarrollar sistemas operativos en tiempo real para microcontroladores. Esto facilita la programación, implementación, protección, conexión y administración de los dispositivos de borde pequeños y de bajo consumo. Es distribuido de forma gratuita con la licencia de código abierta del MIT, incluye un kernel y un conjunto de bibliotecas de software en crecimiento apto para su uso en todos los segmentos y aplicaciones del sector. Se diseñó con un énfasis en la fiabilidad y la facilidad de uso, y ofrece la predictibilidad de las versiones de soporte de largo plazo.

Una ventaja importante es que al ser de código abierto, cuenta con una gran comunidad de usuarios que trabaja para acelerar el desarrollo de soluciones embebidas. La comunidad está activa y existe una lista de correo en la que constantemente se generan discusiones para el mejoramiento del sistema.

Las subrutinas que debe ejecutar la OBC pueden agregarse al conjunto en una estructuras de las llamadas "tareas independientes". Cada tarea se ejecuta dentro de su propio contexto independiente de otras tareas dentro del sistema o del propio programador RTOS. Solo se puede ejecutar una tarea dentro de la aplicación en cualquier momento y el programador RTOS en tiempo real es responsable de decidir qué tarea debe ejecutarse.

El paquete de software incluye una biblioteca de desarrollo basada en Eclipse, una biblioteca con controladores de dispositivos y una herramienta para depurar y cargar software.

Dicho paquete de software está dedicado a inicializar y configurar los puertos de comunicación, funcionando en el sistema operativo FreeRTOS, el cual se muestra a continuación:

```
1
2 /**
3  * NanoMind3
4  *
5  * @author Johan De Claville Christiansen
```

```
 * Copyright 2011 GomSpace ApS. All rights reserved.
 *
 * USING SRAM MEMORY:
 * To store a variable in internal SRAM use:
 * static __attribute__((section(".sram.data")))
 *
 * To place a function in internal SRAM use:
 * __attribute__((section(".sram.text")))
 *
 */

#include <conf_nanomind.h>
#include <conf_io.h>
/* Librerias para puerto serial */
#include <dev/usart.h>
#include <dev/arm/cpu_pm.h>
#include <util/console.h>
#include <util/delay.h>

#include <supervisor/supervisor.h>

#include <csp/csp.h>
#include <csp/interfaces/csp_if_i2c.h>
#include <csp/interfaces/csp_if_kiss.h>

/* Protocolo de comunicaciones */
#if ENABLE_CAN
#include <csp/interfaces/csp_if_can.h>
#endif
#include <csp_extra/csp_console.h>

#include <freertos/FreeRTOS.h>
#include <freertos/task.h>

/* Reloj en tiempo real */
#if ENABLE_RTC
#include <util/clock.h>
#include <dev/arm/ds1302.h>
#endif

/* Librerias standard */
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

/* Agrego libreria CGR */
#include "libcgr.h"
/**********************/

#define F_CPU        40000000
#define F_OSC        8000000
#define F_USART        500000

extern void vTaskServer(void * pvParameters);
extern void vTaskUsartRx(void * pvParameters);
extern void vTaskInit(void *pvParameters);

xTaskHandle handle_server;
xTaskHandle handle_console;
```

```
65
66  int main(void) {
67
68    /* Interfaz Serial - Initialise USART */
69    usart_init(USART_CONSOLE, cpu_core_clk, F_USART);
70
71    /* Initialize delay */
72    delay_init(cpu_core_clk);
73
74    /* La OBC posee un reloj en tiempo real*/
75  #if ENABLE_RTC
76    struct ds1302_clock clock;
77    timestamp_t timestamp;
78
79    /* Get time from RTC */
80    ds1302_init();
81    ds1302_clock_read_burst(&clock);
82    ds1302_clock_to_time((time_t *) &timestamp.tv_sec, &clock);
83    timestamp.tv_nsec = 0;
84    /* Set time in lib-c */
85    clock_set_time(&timestamp);
86  #endif
87    printf("Clock: %d\n", ds1302_get_seconds() );
88    /* Initialize command */
89    command_init();
90
91    /* Initialise CSP */
92    csp_buffer_init(50, 320);
93    csp_init(1);
94
95    /** ROUTING TABLE */
96
97    /*Habiliar el protocolo CAN*/
98  #if ENABLE_CAN
99    /* CAN nodes */
100   struct csp_can_config conf = {.bitrate = 500000, .clock_speed = cpu_core_clk};
101   csp_can_init(CSP_CAN_PROMISC, &conf);
102   csp_route_set(9, &csp_if_can, CSP_NODE_MAC);
103 #endif
104
105   /* KISS nodes */
106   void kiss_putstr_f(char *buf, int len) {
107     usart_putstr(USART_CONSOLE, buf, len);
108   }
109   void kiss_discard_f(char c, void * pxTaskWoken) {
110     usart_insert(USART_CONSOLE, c, pxTaskWoken);
111   }
112   csp_kiss_init(kiss_putstr_f, kiss_discard_f);
113   usart_set_callback(USART_CONSOLE, csp_kiss_rx);
114   csp_route_set(8, &csp_if_kiss, CSP_NODE_MAC);
115
116   /* I2C nodes */
117   csp_i2c_init(1, 0, 400);
118   csp_route_set(2, &csp_if_i2c, CSP_NODE_MAC);
119   csp_route_set(3, &csp_if_i2c, CSP_NODE_MAC);
120   csp_route_set(4, &csp_if_i2c, CSP_NODE_MAC);
121   csp_route_set(5, &csp_if_i2c, CSP_NODE_MAC);
122   csp_route_set(6, &csp_if_i2c, CSP_NODE_MAC);
123   csp_route_set(7, &csp_if_i2c, CSP_NODE_MAC);
```

```
124
125   /* Default route */
126   csp_route_set(CSP_DEFAULT_ROUTE, &csp_if_i2c, 5);
127
128   /* Start router */
129   csp_route_start_task(1024*4, 3);
130
131   /* Initialise Console */
132   console_init();
133   console_set_hostname(CONFIG_HOSTNAME);
134
135   /* Habilitacion protocolo CSP*/
136 #ifdef ENABLE_CSP_CLIENT
137   csp_console_init();
138   csp_set_hostname(CONFIG_HOSTNAME);
139   csp_set_model(CONFIG_MODEL);
140 #endif
141
142 #if ENABLE_SUPERVISOR
143   /* Start supervisor */
144   sv_init(1000);
145 #endif
146
147 #if ENABLE_CPP
148   /* C++ static constructors */
149   extern void (*__init_array_start []) (void) __attribute__((weak));
150   extern void (*__init_array_end []) (void) __attribute__((weak));
151   int count = __init_array_end - __init_array_start, i;
152   for (i = 0; i < count; i++)
153     __init_array_start[i]();
154 #endif
155
156   /* Inicio de las tareas - Start tasks*/
157   xTaskCreate(debug_console, (const signed char *) "CONSOLE",
158                                 1024*4,
159                               NULL,
160                                 1,
161                           &handle_console);
162   xTaskCreate(vTaskInit, (const signed char *) "INIT",
163                            1024*4,
164                              NULL,
165                                 1,
166                              NULL);
167   xTaskCreate(vTaskServer,     (const signed char *) "SRV",
168                                1024*4,
169                                  NULL,
170                                     1,
171                            &handle_server);
172   xTaskCreate(vTaskUsartRx, (const signed char*) "USART",
173                              1024,
174                              NULL,
175                                1,
176                               NULL);
177   /*****************************/
178   /* Inicio tarea CGR en FreeRTOS */
179   xTaskCreate(cgr, (const signed char*) "cgr",1024*4 ,
180                              NULL,
181                                 1,
182                             NULL);
```

```
183   /*****************************/
184
185   /* Timer uses LFCLOCK = F_OSC/2 */
186   vTaskStartScheduler(F_OSC/2, 1024*4);
187
188   /* Should never reach here */
189   while(1) exit(0);
190
191 }
```

Código 3.1 – *Rutina principal main.c en FreeRTOS*

3.1.7. Manejo de Memoria en FreeRTOS

El kernel RTOS necesita RAM cada vez que se crea una tarea, una cola, una exclusión mutua, un temporizador de software, un semáforo o un grupo de eventos. La memoria RAM se puede asignar automáticamente de forma dinámica desde el *Heap* de FreeRTOS dentro de las funciones de creación de objetos de la API de RTOS, o puede ser asignada manualmente en el programa desarrollado como es el caso.

Las funciones malloc() y free() de la biblioteca C estándar se pueden usar para asignar memoria dinámicamente, pero no siempre están disponibles en los sistemas embebidos, ocupan mucho espacio de código, no son seguros para subproceso y no son deterministas (la cantidad de tiempo necesario para ejecutar la función diferirá de una llamada a otra). Entonces es más probable que se utilice una subrutina alternativa de asignación de memoria.

FreeRTOS mantiene la API de asignación de memoria en su capa portátil. La capa portátil está fuera de los archivos de origen que implementan la funcionalidad principal de RTOS, lo que permite proporcionar una implementación específica de la aplicación apropiada para el sistema en tiempo real que se está desarrollando.

Cuando el kernel RTOS requiere RAM, en lugar de llamar a malloc(), llama a pvPortMalloc(). Cuando se libera RAM, en lugar de llamar a free(), el kernel RTOS llama a vPortFree()[121].

Heap_3.c utiliza las funciones malloc() y free() de la biblioteca estándar, por lo que el tamaño del heap está definido por la configuración del linker y la configuración de configTOTAL_HEAP_SIZE no tiene ningún efecto.

Heap_3 hace que malloc() y free() sean seguros para subprocesos al suspender temporalmente el programador de FreeRTOS. La seguridad de subprocesos y la suspensión del programador son temas que se tratan en el Capítulo 7, Administración de recursos.

3.2. Implementación del CGR en la NanoMind

El algoritmo presente en ION (del inglés, *Interplanetary Overlay Network*) fue estudiado con el objetivo de tomar la subrutina donde se ejecuta CGR, que calcula las rutas desde un nodo a otro. También se han adaptado las subrutinas que realizan lecturas de datos, de modo a que no necesite otras funciones específicas de ION para correr.

Para la implementación de CGR en la NanoMind, se realizó la revisión del código y se extrajeron las funciones principales con ayuda de [32] y [38], donde se explican en detalle el funcionamiento de cada subrutina.

El algoritmo de ION ha servido como base para estructurar una nueva implementación, y debido a su complejidad, se han creado funciones propias para la utilización del plan de

contactos en estructuras de datos y listas enlazadas, la determinación de rutas con el menor "costo" a destino (en este caso corresponde a un menor tiempo de llegada) mediante el algoritmo de Dijkstra [114], y la selección de los siguientes nodos para la transferencia de datos. Como resultado, el algoritmo de Dijkstra encuentra todas las rutas posibles desde un nodo a otro. Al encontrar las rutas, se determinan cuáles podrían ser los siguientes nodos que el paquete, que tienen una ruta válida a destino. Por último se decide cuál de todos esos nodos recibirá definitivamente el paquete, teniendo en cuenta parámetros de nodos que estén excluidos, nodos que no responden, etc.

3.2.1. Creación de la Tarea CGR

Massachusetts Institute of Technology

La función para crear tareas en FreeRTOS es la siguiente:

```
1 xTaskCreate( TaskFunction_t pvTaskCode,
2              const char * const pcName,
3              unsigned short usStackDepth,
4              void *pvParameters,
5              UBaseType_t uxPriority,
6              TaskHandle_t *pxCreatedTask)
7
```

Código 3.2 – *Función para crear una tarea*

que es la más compleja de FreeRTOS [48].

- El primer parámetro es simplemente un puntero a la función que implementa la tarea ("cgr" en este caso).
- El segundo parámetro es un nombre descriptivo para la tarea, para que pueda identificarse fácilmente.
- El tercer parámetro es el tamaño del *"stack"* en la memoria que será asignado por el kernel. Este valor especifica la cantidad de palabras que el *"stack"* puede almacenar.
- El cuarto parámetro es el valor que puede ser pasado a la tarea que se está creando. Es de un tipo puntero a *void* (void*), de esta manera y al hacer un *casting* permite enviar y recibir un parámetro de cualquier tipo al "castear" de vuelta el puntero al tipo de dato que sea necesario en la definición de función.
- El quinto parámetro define la prioridad con la que la tarea se va a ejecutar, donde el nivel más bajo de prioridad es cero y la mas alta está definida en la constante (configMAX_PRIORITIES – 1)
- El sexto parámetro es usado para referenciar la tarea que está siendo creada, en una llamada a una función que por ejemplo reasigne la prioridad de la función.

La tarea CGR se carga en el conjunto de tareas de la OBC en la subrutina principal 3.1 en la linea 179 mediante esta función API:

```
1
2  /* Creo tarea CGR en FreeRTOS */
3  xTaskCreate(cgr, (const signed char*) "cgr",1024*4 , NULL, 1, NULL);
4
5  /*****************************/
```

Código 3.3 – *Creación de tarea CGR en FreeRTOS*

En este punto se pueden agregar más tareas correspondientes a otras funciones de control, comando o manejo de datos del satélite, pero por el enfoque dado al trabajo de investigación, se procede a utilizar todos los recursos de la OBC en la única tarea de interés.

3.2.2. Estructura

Para comparar la implementación hecha, se muestran las diferencias (tipos de datos, estructura) que existen entre la subrutina de ION y el código adaptado para correr en la *Nanomind* OBC para nanosatélites.

```
/* codigo extraido directamente de ION */
typedef struct
  { /* Working values, reset for each Dijkstra run.*/

    IonCXref  *predecessor; /* On path to destination. */
    time_t    arrivalTime;  /* As from time(2). */
    int   visited;  /* Boolean. */
    int   suppressed; /* Boolean. */
  } CgrContactNote; /* IonCXref routingObject is one of these. */


```

Código 3.4 – *Estructrura de datos en ION*

A continuación se muestra la sencilla definición de la función principal del código implementado:

```

  /*Subrutina principal*/
  void cgrForward(Bundle *bundle, Contact *cp, nodeList *En);
```

Código 3.5 – *Funcion principal del algoritmo implementado*

donde:

- bundle, es el paquete de datos a enviar, se define el nodo fuente y destino
- cp, es el plan de contactos contenida en una lista enlazada
- En, son los nodos excluidos, que pueden estar inhabilitados por razones comerciales, funcionales, etc.

en comparación a la definición de la misma función en ION:

```
  static int computeDistanceToTerminus(IonCXref *rootContact,
   CgrContactNote *rootWork,
   IonNode *terminusNode,
   time_t currentTime,
   PsmAddress excludedEdges,
   CgrRoute *route,
   CgrTrace *trace );
```

Código 3.6 – *Funcion del mismo algoritmo en ION*

En la última subrutina los tipos de datos están descritos en otros archivos, siendo parte de un proyecto más grande. Esto ocasiona que a primera vista se pierda la perspectiva de la estructura principal del código. Sin un meticuloso seguimiento es complicado determinar los parámetros principales dentro de las estructuras de datos. Para empezar la arquitectura del código, primero se deben definir los principales tipos de datos con los que se van a trabajar.

Entonces, a continuación y a modo de ejemplo, se muestra la estructura de un contacto utilizada en ION 3.7.

Luego se muestra el tipo de estructura "CgrRoute" donde se describen los tipos de datos: PsmAddress (linea 6), uvast (linea 16), Scalar (30-31). Estos tipos de datos están implementados en diferentes librerías, distribuidas en el código ION.

```
1  `
2    /* Functions for managing the CGR database. */
3
4    typedef struct
5    {
6      PsmAddress  rootOfSpur; /* Within *prior* route. */
7      int    spursComputed;  /* Boolean. */
8
9      /* Address of list element referencing this route, in
10     either a knownRoutes list or a selectedRoutes list. */
11
12     PsmAddress  referenceElt;
13
14     /* Contact that forms the initial hop of the route. */
15
16     uvast    toNodeNbr;  /* Initial-hop neighbor. */
17     time_t     fromTime; /* As from time(2). */
18
19     /* Time at which route shuts down: earliest contact
20     end time among all contacts in the end-to-end path. */
21
22     time_t     toTime;   /* As from time(2). */
23
24     /*Detalles de la Ruta - Details of the route. */
25
26     float    arrivalConfidence;
27     time_t     arrivalTime;  /* Earliest arrival time. */
28     PsmAddress  hops;   /* SM list: IonCXref addr. */
29
30     Scalar  overbooked; /*Bytes needing reforward*/
31     Scalar  protected;  /*Bytes not overbooked  */
32     double  maxVolumeAvbl;
33     size_t  bundleECCC;
34     time_t  eto;     /*Earliest xmit oppor'ty*/
35     time_t  pbat;    /*Proj. bundle arr. time*/
36
37
38   } CgrRoute;
39
40
```

Código 3.7 – *Estructura de una Ruta en ION*

en comparación a la estructura 3.9 de la nueva implementación que se realizó, que se muestra a continuación.

3.2.3. Estructura Principal

Para empezar a describir la estructura del algoritmo, primero se debe mencionar los tipos de datos utilizados.

Se tiene "*node*" como tipo de dato básico, definido como un "*unsigned int*".

```
1
2  typedef unsigned int node;
3
```

Mediante este tipo de dato básico se construye una lista enlazada que guarde una lista de nodos:

```
1  struct node_list {
2
3    node nInL;        //lista de nodos
4    struct node_list *next;
5
6  };
```

Código 3.8 – *Lista enlazada básica*

Y esta estructura de datos se utiliza en la estructura específica "*contact*" para guardar la lista de nodos visitados, lista que es usada por el algoritmo de Dijkstra durante la búsqueda de rutas.

La estructura de datos "*contact*" utilizada se muestra en la figura 3.9, donde se puede identificar tres áreas de datos. La primera (lineas 4-8)es información tomada de un plan de contactos; la segunda (lineas 11-18), un área de trabajo, la tercera(linea 26), un área de almacenamiento de variables que utiliza el algoritmo de Dijkstra como banderas.

```
1  struct contact{
2
3    /* parametros minimos para UN contacto ISL */
4    node source;
5    node destination;
6    time_t start;
7    time_t end;
8    unsigned int rate;
9
10   //area de trabajo (utilizada por Dijkstra)
11   time_t arrivalTime;         //tiempo de llegada
12   unsigned int capacity;      //capacidad del canal
13   unsigned int residualCapacity;
14   unsigned int confidence;    //confiabilidad
15   int visited;            //boolean
16   int suppressed;             //contacto suprimido
17   struct contact *predecessor; //guardar el contacto anterior
18   struct contact *sucessor;   //guardar el contacto siguiente
19
20   /* El algoritmo de Dijkstra
21   * toma en cuenta los nodos ya visitados
22   * por lo que se debe dejar un registro
23   */
24
25   /* Para almacenar los nodos ya visitados */
26   nodeList *visitedNodes;
27
28   /* Cabeceras de las listas */
29   struct contact *next;
30   struct contact *prev;
31
32 };
33
34 typedef struct contact Contact;
```

```
35
```

Código 3.9 – *Estructura de datos implementada*

Esta estructura es de mayor sencillez que la mostrada en el código de ION. No depende de las funciones que allí se utilizan y puede correr independientemente de ellas. Las variables de trabajo en ION se definen de la siguiente manera: el contacto anterior, el tiempo de llegada del dato, si ya ha sido "visitado" o si ha sido "suprimido" (en casos de que este nodo esté imposibilitado de enviar o recibir paquetes) en las lineas 5-8.

```
1  /*codigo extraido directamente de ION */
2   typedef struct
3   { /*Working values , reset for each Dijkstra run . */
4
5     IonCXref *predecessor ; /*On path to destination . */
6     time_t arrivalTime ; /*As from time(2) . */
7     int v i s i t e d ; /*Boolean . */
8     int suppressed ; /*Boolean . */
9     } CgrContactNote ; /*IonCXref routingObject is one of these . */
10
```

Código 3.10 – *Estructrura de datos en ION*

En cuanto al dato o paquete de dato que se desea enviar, se tiene la estructura:

```
1  typedef struct {
2
3    char *data;
4    unsigned int bitlength;
5
6    time_t startTime;
7    time_t deadline;
8    node startnode;
9    node destination; // no confundir destino de contacto con destino de bundle
10
11   // metricas para tomar futuras decisiones
12   int critical;
13
14 }Bundle;
```

Código 3.11 – *Estructura de datos que almacena los parámetros asociados al paquete de datos a enviar*

Donde se establecen los parámetros básicos de los paquetes de datos a enviar(lineas 3-8), como es la llamada carga útil, el tamaño de la palabra en bits, el momento en que se genera el dato, cuándo el dato deja de ser relevante, en qué nodo se genera el dato y el destino final del dato.

Para el manejo de las rutas (lista de saltos entre nodos) que debe seguir el paquete de datos, se tiene como datos asociados a los contactos, el siguiente contacto, la cantidad de saltos, el último momento de transmisión, el tiempo de llegada y la capacidad del enlace.

Además de esto, al tener varias rutas distintas, se debe poder listar estas rutas para luego elegir la mejor, por esto se tiene la estructura "*routeList*" que almacena la lista de rutas.

```
1  struct routeStr{
2
3    Contact *ctcList;
4    node firstHop;
5    nodeList *hops;
6    unsigned int hopCount;
7    time_t toTime;
```

```
8     time_t arrivalTime;
9     unsigned int capacity;
10
11  };
12
13
14  struct routeList{
15
16    struct routeStr *route;
17    //struct routeList *prev;
18    struct routeList *next;
19  };
```

Código 3.12 – *Estructura de datos para la ruta de contactos*

Se muestran también las funciones a ser utilizadas.

Existe una jerarquía en estas funciones, donde una función llama a otra. La función de mayor jerarquía es la "*cgrForward*" 3.17, seguidas por las rutinas de selección de rutas, que llaman a la función "*dijkstra*" y luego las de menor jerarquía que manejan las búsquedas y adición de nodos en las listas enlazadas.

```
1
2   /*Subrutina principal*/
3   void cgrForward(Bundle *bundle, Contact *cp, nodeList *En);
4
5   /* rutinas de manejo de listas */
6   // agregar nodos a una lista
7   nodeList *addToNodeList(nodeList *, node);
8   prxnodeList *addToPrxNodeList(prxnodeList *,prxnode);
9   struct routeList *appendRoute(struct routeList *, struct routeStr *);
10  //rutinas de busqueda en lista
11  unsigned int isInNodeList(nodeList *, node);
12  unsigned int isInPrxNodeList(prxnodeList *, node);
13
14  //Algoritmos de seleccion de rutas
15  struct routeList *cgrAllPathsAnchor(node, node, time_t, Contact *);
16  struct routeList *first_depleted(node, node, time_t , Contact *, int);
17
18  //Para limpieza de las variables de trabajo
19  void clearWorkAreas(Contact *);
20  void clearWorkArea(Contact *);
21
22  //Para crear el dato
23  Bundle *create_bundle(node, node);
24  struct contactList *createNeighborList(Contact *cp);
25
26  //Para busqueda de la mejor RUTA
27  struct routeStr *dijkstra(Contact *, node , Contact *);
28
29  //Explorar rutas y encontrar nodos siguientes
30  struct routeList *loadRouteList(Bundle *, Contact *);
31  prxnodeList *identifyProximateNodes(Bundle *, Contact *, nodeList *);
32  time_t min(time_t, time_t);
33
34  //para determinar los contactos alternativos o vecinos
35  //fuente de contacto = fuente del suiguiente contacto, (ruta alternativa)
36  Contact *nextNeighbor(Contact *cp);
37
38  //para determinar el proximo contacto del destino con la siguiente fuente
```

```
39  //destino = siguiente fuente (la misma ruta pero siguiente salto)
40  Contact *nextContact(node, Contact *cp);
41
42  //impresion de rutas
43  void printRoute(struct routeStr *);
44  void printRoutes(struct routeList *);
45  void printCtc(Contact *);
46  void printCp(Contact *);
47
48  //contadores de memoria
49  void memCount(size_t);
50  void memFreedCount(size_t);
```

Código 3.13 – *Lista de Funciones Utilizadas*

Y la función que crea el paquete de prueba 3.14. En este paquete se inserta el dato (Línea 15), teniendo en cuenta el "*deadline*" (Línea 19) para decidir en un futuro cuando el paquete deja de ser relevante y la longitud del mismo. También se tiene en cuenta la criticidad del paquete, valorándose en la variable "*critical*"(Línea 21), como puede verse:

```
1   Bundle *create_bundle(node  source, node dest){
2
3     Bundle *bundle;
4
5     /* Creo una cadena de caracteres a enviar */
6     char data[] = "CGR_IS_THE_FUTURE";
7
8     /* Asigno memoriaMassachusetts Institute of Technology para el paquete */
9     bundle=pvPortMalloc(sizeof(Bundle));
10    bundle->data     = pvPortMalloc(sizeof(data));
11    if(bundle->data == NULL)
12      printf("pvPortMalloc in createBundle fail");
13
14    /* Cargo parametros necesarios */
15    bundle->data     = data;
16    bundle->startnode = source;
17    bundle->destination = dest;
18    bundle->startTime = 100;
19    bundle->deadline  = 100000;
20    bundle->bitlength    = sizeof(bundle->data)*8;
21    bundle->critical = 0;
22
23    return bundle;// envio solo la direccion de memoria
24  }
```

Código 3.14 – *Subrutina que crea los datos y parámetros a enviar*

Además de reimplementar CGR se debe poder realizar experimentos con la OBC, que utiliza FreeRTOS para gestionar las tareas, por lo tanto, antes de cada experimento, se prepara el código con las variables de prueba: Tiempo de ejecución ($tiempoFinal - tiempoInicial$), contador de memoria utilizada y contador de memoria liberada.

```
1  clock_t start, stop;
2  totalMem = 0;
3  freedMem = 0;
```

Código 3.15 – *Variables del experimento*

A continuación se muestra la estructura de la subrutina principal CGR en el código 3.16.

La OBC se programa de modo que la subrutina CGR se ejecute una vez con los parámetros y paquete a enviar. Manualmente, se varía el nodo destino del paquete "*Bundle*" (Línea 11). Otros parámetros como la cantidad de contactos, y nodos excluidos ya dados se definen también (Líneas 14-16).

Mediante un ciclo "*for*" se varía incrementalmente la cantidad de contactos a cargar (Línea 22) en cada corrida del algoritmo. Esto es así por la necesidad de medir los tiempos de ejecución en función a la cantidad de contactos cargados 4.1.

Luego de iniciar los contadores de parámetros principales (Línea 25, 26), se carga el plan de contactos con la cantidad de nodos *i* (Línea 31) y comienza el conteo del tiempo de ejecución mediante la función *"xTaskGetTickCount()"* (Línea 34) que devuelve el tiempo desde que el planificador de tareas*"vTaskStartScheduler"* del sistema operativo FreeRTOS ha sido llamado.

A continuación se ejecuta la rutina *"cgrForward()"* (Línea 37), y seguidamente se vuelve a llamar a la función *"xTaskGetTickCount()"*. La diferencia entre los valores (Línea 43) devueltos es lo que se considera el tiempo de ejecución total del algoritmo.

Seguidamente se imprimen los resultados en pantalla (lineas 43-46) y se libera la memoria (linea 48-58) para la siguiente iteración con mayor número de contactos.

```
1
2  #include "cgr.h"
3
4  void cgr (void *pvParameters) {
5
6    //Se crea las variables de almacenamiento de listas de contactos
7    Contact *cp, *cpaux;
8
9    //Se crea el supuesto dato de prueba con parametros de fuente y destino
10   Bundle *bundleP;
11   bundleP=create_bundle(15, 117);
12
13   //Se agrega los nodos que debe excluir
14   nodeList *En=NULL;
15   node exNode = 5;
16   En = addToNodeList(En, exNode);
17
18
19   //Para ejecutar la busqueda variando la cantidad de contactos agregados
20   int cantMaxNodos = 1200;
21   int i;
22   for(i=100 ; i<= cantMaxNodos; i+=100){
23
24     //Inicializo contadores de memoria
25     totalMem = 0;
26     freedMem = 0;
27
28     printf("En %d nodos:",i);
29
30     // cargo el plan
31     cp=load_plan(i);  //printf("Plan cargado\n");
32
33     //start counter empiezo el conteo
34     portTickType start = xTaskGetTickCount();
35
36     //start CGR empieza la subrutina
37     cgrForward(bundleP, cp, En);
38
```

```
39      //stop counter
40      portTickType stop = xTaskGetTickCount();
41
42      //imprimo resultados
43      printf("El tiempo es:\t%lu\n",stop-start);
44      printf("memoria utilizada:\t%d\n", totalMem);
45      printf("memoria liberada:\t%d\n", freedMem);
46      fflush(stdin);
47
48      //libero toda la memoria
49      cpaux = cp;
50      for( ;cpaux != NULL; ){
51        cp = cpaux;
52        cpaux=cpaux->next;
53        if(!cp->sucessor) vPortFree(cp->sucessor);
54        if(!cp->predecessor) vPortFree(cp->predecessor);
55        if(!cp->visitedNodes) vPortFree(cp->visitedNodes);
56        vPortFree(cp);
57      }
58    }
59    //borro la tarea una vez terminada
60    vTaskDelete(NULL);
61
62  }
```

Código 3.16 – *Subrutina principal*

3.2.4. Estructura de *cgrForward*

A diferencia de las rutas de Internet, las rutas DTN se expresan en función del tiempo. Por lo tanto, solo son válidos por un período de tiempo determinado y CGR debe revisarlos para cada paquete nuevo.

La rutina de reenvío CGR representada en el algoritmo 3.16 (Línea 37), 3.17 se llama en cualquier nodo de la red cada vez que se va a reenviar un nuevo paquete B. Inicialmente, el algoritmo recibe la vista local de la topología expresada en el plan de contactos Cp actualizada desde la última llamada (Línea 1).

Mediante la función "*identifyProximateNodes()*" 3.18 se crea una estructura de lista de rutas Rl, que contiene todas las rutas válidas a cada destino conocido, para forzar una actualización de la tabla de rutas. De hecho, la lista de rutas Rl se deriva del plan de contacto local Cp (Línea 5).

A continuación, el procedimiento completa una lista de nodos próximos Pn que comprende todos los nodos posibles que, de acuerdo con la lista de rutas Rl, tienen una ruta válida hacia el destino (Algoritmo 3.18).

Este paso es ejecutado por la rutina "*identifyProximateNodes*" que se detalla continuación. En este paso se recibe una lista de nodos excluidos En por referencia, para evitar la consideración de vecinos administrativamente prohibidos (por ejemplo, nodos que no responden) o el remitente del paquete anterior para minimizar los bucles de ruta.

Si el paquete es crítico, se puede clonar el paquete a todos los nodos de la lista 3.18 (Línea 15), si no es crítica, se elije un solo nodo de toda la lista. Los nodos vecinos con mejor tiempo de llegada son la primera prioridad (Líneas 24-50), en segundo nivel de prioridad, los que tienen menor cantidad de saltos entre nodos (Líneas 37-39) y en tercer lugar, los nodos con menor identificación (Líneas 45-47).

Luego, si se encuentra un nodo apropiado el paquete "*Bundle*" es puesto en cola para envío (Línea 54).

Cuando la rutina CGR termina, el paquete puede enviarse o no, de acuerdo a las prioridades de otros procesos de mayor nivel. La discusión de tales procesos escapan al alcance de este trabajo.

```
1  void cgrForward(Bundle *bundle, Contact *cp, nodeList *En){
2
3    prxnode nextHop;
4    prxnodeList *Pn, *Pnaux;
5    Pn=identifyProximateNodes(bundle, cp, En);
6    /*
7    printf("Proximate nodes: ");
8    for(prxnodeList *prxn = Pn; prxn!=NULL;prxn=prxn->next)
9    {
10     printf(" %d,",prxn->prxmNode.node);
11
12   }
13   printf("\n");
14   */
15   if(bundle->critical) //enqueue a copy of B to each node in Pn
16
17   return ;
18
19   nextHop.node = 0;
20   nextHop.hopCount =0;
21   nextHop.arrivalTime = ULONG_MAX;
22   nextHop.id = 0;
23
24   for(prxnodeList *prxn = Pn; prxn!=NULL;prxn=prxn->next)
25   {
26     if (nextHop.node == 0) {
27       nextHop = prxn->prxmNode;
28
29     }else if(prxn->prxmNode.arrivalTime < nextHop.arrivalTime) {
30
31       nextHop = prxn->prxmNode;
32
33     }else if(prxn->prxmNode.arrivalTime > nextHop.arrivalTime)
34
35     continue;
36
37     else if(prxn->prxmNode.hopCount < nextHop.hopCount) {
38
39       nextHop = prxn->prxmNode;
40
41     }else if(prxn->prxmNode.hopCount > nextHop.hopCount)
42
43     continue;
44
45     else if(prxn->prxmNode.id < nextHop.id){
46
47       nextHop = prxn->prxmNode;
48     }
49
50   }
51
52   if (nextHop.node != 0) {
```

```
53
54      printf("Next hop : %u,",nextHop.node);
55    }
56    printf("\n");
57    fflush(stdin);
58
59    Pnaux = Pn;
60    for(;Pnaux != NULL; ){
61      Pn = Pnaux;
62      Pnaux = Pn->next;
63      vPortFree(Pn);
64    }
65
66  }
```

Código 3.17 – *Función principal de selección de nodos*

3.2.5. Estructura de Identificación de Nodos

Teniendo el paquete "*Bundle*", el Plan de Contactos y la lista de nodos excluidos, se procede a llenar una lista de posibles nodos que, de acuerdo a una lista de rutas *Rl* tienen un recorrido válido hacia el nodo destino. Este procedimiento se ejecuta en la rutina "*ProximateNodeList*" 3.18.

Cada una de estas rutas necesita ser evaluada para sumar posibles futuros nodos a lista Pn. Esta lista es usada por "*cgrForward*" 3.17 y está formada por un conjunto de nodos vecinos no repetidos que son capaces de alcanzar el destino del paquete "*Bundle*". Si la lista de rutas RL está vacía, la función "*loadRouteList(Bundle, cp)*" 3.19 es llamada (Línea 11) para encontrar todas las rutas hacia el destino del paquete B. Las rutas que no satisfacen cierto criterio son descartadas (Líneas 15-26).

En particular, las rutas con el último momento de transmisión en el pasado (ya vencida) (Línea 17), un tiempo de llegada posterior al tiempo límite del paquete (Línea 19), una capacidad inferior al tamaño del paquete (Línea 21) y un nodo próximo dentro de los nodos excluidos (Línea 23), o que requieren una cola de salida local que se agota (Línea 25), se ignoran. Las rutas restantes se consideran adecuadas y el nodo próximo correspondiente en la lista Pn se reemplaza por una ruta mejor (Algoritmo 3.18, líneas (27-51)) o se agrega directamente a la lista (Algoritmo 3.18, líneas (54-61)).

El criterio de reemplazo es coherente con el Algoritmo 3.17: primero se considera el mejor tiempo de llegada y luego el conteo de saltos de la ruta. Durante este proceso, las métricas de ruta necesarias, como el tiempo de llegada y el número de saltos de la mejor ruta, también se almacenan en la estructura de datos de cada nodo próximo contenida en Pn.

```
1
2   prxnodeList *identifyProximateNodes(Bundle *bundle, Contact *cp, nodeList *En){
3
4     prxnodeList *Pn=NULL; //Proximate node list
5     prxnode pn;
6     prxnodeList *nl;
7     struct routeList *rtl, *Rl=NULL;
8
9     time_t currTime = 1000;
10
11    if(!Rl) Rl=loadRouteList(bundle,cp);
12    printRoutes(Rl);
```

```
13
14
15    for(rtl=Rl;rtl!=NULL;rtl=rtl->next){
16      //ignore past route
17      if(rtl->route->toTime <= currTime)          continue;
18      //route arrives late
19      if(rtl->route->arrivalTime >= bundle->deadline)   continue;
20      //not enough capacity
21      if(rtl->route->capacity < bundle->bitlength)    continue;
22      //next hop is excluded
23      if( isInNodeList(En, rtl->route->ctcList->destination))     continue;
24      //outbound queue depleted
25      if((rtl->route->capacity) < bundle->bitlength)          continue;
26      //printRoute(rtl->route);
27      for(nl=Pn;(nl!= NULL);nl=nl->next){
28
29        if(nl->prxmNode.node == rtl->route->ctcList->destination){
30
31
32          if(nl->prxmNode.arrivalTime > rtl->route->arrivalTime){
33            nl->prxmNode.node = rtl->route->ctcList->destination;
34            nl->prxmNode.arrivalTime = rtl->route->ctcList->arrivalTime;
35          }
36          //previous route was better
37          else if(nl->prxmNode.arrivalTime<rtl->route->arrivalTime){
38            continue;
39          }
40
41          else if(nl->prxmNode.hopCount > rtl->route->hopCount){
42            //previous route was better
43            continue;
44          }
45
46          else if(nl->prxmNode.hopCount < rtl->route->hopCount){
47            continue;
48            //previous route was better
49          }
50          break;
51        }
52      }
53      //if route.nextHop does not belong to Pn
54      if(!isInPrxNodeList(Pn,rtl->route->ctcList->destination)){
55        //printf("%u,",Pn->prxmNode.node);
56        pn.node = rtl->route->ctcList->destination;
57        pn.arrivalTime = rtl->route->ctcList->arrivalTime;
58        pn.hopCount = rtl->route->hopCount;
59        Pn = addToPrxNodeList(Pn, pn);
60
61      }
62
63    }
64
65    rtl = Rl;
66    for(;rtl != NULL ;){
67      Rl=rtl;
68      rtl = Rl->next;
69      vPortFree(Rl->route->hops);
70      vPortFree(Rl);
71
```

```
72      }
73      return Pn;
74    }
75
```

Código 3.18 – *Función de Identificación de Nodos*

3.2.6. Estructura de Carga de Rutas de Paquete *loadRouteList*

Esta función selecciona el algoritmo que formará las rutas en versiones anteriores de ION se utilizaba un mecanismo de anclaje "*anchor*" que fue reemplazada por un algoritmo que elije las rutas de acuerdo a capacidad o el volumen de datos que puede manejar ("*First Depleted*") (Línea 9) pero con esta rutina se da la opción de seguir agregando mejores mecanismos de búsqueda de rutas.

```
1
2 struct routeList *loadRouteList(Bundle *bundle, Contact *cp){
3
4   struct routeList *Rl;
5   //node, node, time_t, Contact *
6   // version anterior de selec. de rutas
7   //Rl = cgrAllPathsAnchor(2, 6, 3000, cp);
8   //version actual de selec. de rutas
9   Rl = first_depleted(bundle->startnode, bundle->destination,
10    bundle->startTime, cp, 1);
11
12  return Rl;
13 }
```

Código 3.19 – *Función de carga de rutas*

3.2.7. Estructura de Selección *firstDepleted*

Para encontrar todas las rutas posibles en el plan de contactos, la rutina de carga de la lista de rutas realiza una serie de búsquedas de Dijkstra en un gráfico de contactos derivado del plan de contactos.

Se reserva un área de trabajo para cada contacto en el plan de contactos para ejecutar las búsquedas de Dijkstra.

Inicialmente, el algoritmo crea un contacto raíz "*rootContact*" (Líneas 9-25) y también borra todos los parámetros necesarios en cada área de trabajo de los contactos (Algoritmo 3.20, líneas (27-37).

A continuación, la rutina realiza un bucle (Líneas 42-56) para encontrar diferentes rutas utilizando el algoritmo de Dijkstra (Algoritmo 3.20, (línea 44)). Si el algoritmo de Dijkstra no encuentra una ruta válida, el ciclo termina (Línea 45).La métrica que impulsa la búsqueda del camino más corto es el tiempo de llegada, que debe calcularse a partir de la hora de inicio y el retraso previsto de cada contacto en el camino explorado. El ciclo *for* visita cada contacto para realizar los calculos de agotamiento de capacidad de volumen de modo que el primero en agotarse es el que será suprimido (Líneas 47-54). Para garantizar que cada ejecución de Dijkstra proporciona una ruta distinta, el proceso de carga de la lista de rutas elimina el contacto suprimido de la última ruta encontrada de la siguiente búsqueda (Línea 51). El algoritmo de Dijkstra se encarga de no utilizar ese contacto suprimido en la siguiente iteración.

Aunque se pueden encontrar múltiples rutas utilizando contacto inicial + anclaje, sólo se pueden utilizar las rutas con capacidad residual suficiente para transmitir los paquetes. Los recursos de volumen de la mejor ruta son utilizados por los paquetes iniciales. Por lo tanto, para minimizar los cálculos de ruta innecesarios, se considera el volumen residual para calcular las mejores rutas (primero agotadas) en el ciclo *for* (Líneas 47-54).

El contacto con el volumen más corto se suprime primero y el volumen residual para todos los saltos (contactos) en la ruta se calculan como: $Volumenresidual = Volumentotalactual - Volumendelcontactomsbajo$ (Línea 48) Al encontrarse una posible ruta, esta se va registrando en una lista "*routes*" (Línea 55) que será el resultado final del algoritmo. La búsqueda finaliza cuando no se encuentran más rutas en el grafo de contactos.

```
1  struct routeList *first_depleted(node src, node dest, time_t currTime,
2  Contact *cp, int keep_residual_capacity){
3
4    int residualCapacity;
5    struct routeStr *Rt=NULL;
6    struct routeList *routes=NULL;
7    Contact *curCtc, *rootCtc;
8    curCtc = rootCtc = NULL;
9    // create rootcontact
10   rootCtc = pvPortMalloc(sizeof(Contact));
11   if(rootCtc == NULL){
12     printf("pvPortMalloc in first_depleted fail");
13   }
14   rootCtc->source = src;
15   rootCtc->destination = src;
16   rootCtc->start = 0;
17   rootCtc->end = LONG_MAX;
18   rootCtc->confidence =1;
19   rootCtc->rate = 100;
20   rootCtc->capacity = rootCtc->rate*(rootCtc->start - rootCtc->end);
21   rootCtc->predecessor = NULL;
22   rootCtc->sucessor  = NULL;
23   rootCtc->next     = NULL;
24   rootCtc->arrivalTime = currTime;
25   rootCtc->visitedNodes= NULL;
26
27   //reset suppressed contacts
28   curCtc=cp;
29   while(curCtc!=NULL){
30     curCtc->residualCapacity= curCtc->capacity;
31     residualCapacity = curCtc->residualCapacity;
32     clearWorkArea(curCtc);
33     curCtc->suppressed = FALSE;
34
35     if(keep_residual_capacity){
36       curCtc->residualCapacity = residualCapacity;
37     }
38     curCtc=curCtc->next;
39   }
40
41
42   while(TRUE){
43
44     Rt = dijkstra(rootCtc, dest, cp);
45     if(!Rt) break;
46     //consume volume in all hops and supress limiting hop
```

```
47      for(curCtc = Rt->ctcList;curCtc != NULL; curCtc = curCtc->next){
48        curCtc->residualCapacity -= Rt->capacity;
49
50        if(curCtc->residualCapacity == 0){
51          curCtc->suppressed = TRUE;
52
53        }
54      }
55      routes = appendRoute(routes, Rt);
56    }
57    vPortFree(rootCtc);
58    return routes;
59
60  }
61
```

Código 3.20 – *Función firstDepleted "primero en agotarse"*

3.2.8. Estructura Algoritmo de Dijkstra

El algoritmo de búsqueda Dijkstra, encuentra la mejor ruta a través de un grafo de contacto se presenta en 1. El resultado final del procedimiento es una ruta R_S^D desde el origen S al destino D, con un tiempo de llegada BDT. La ruta se calcula partiendo del contacto raíz ($root$) hasta el destino $dest$ en el plan de contactos $plan$.

El contacto raíz $root$ es un contacto artificial $C_{S,S}^{0,\infty}$ desde el origen S a sí mismo.

La hora de llegada al contacto raíz $C_{S,S}^{0,\infty}.arr\_time$ se establece en la hora de inicio de la ruta esperada (es decir, la hora a la que se llama a la búsqueda de Dijkstra, para el ejercicio este tiempo es el momento de inicio del paquete, $bundle->startTime$, en 3.19, línea 10). Por tanto, la búsqueda de ruta ignorará los contactos que terminen antes de $C_{S,S}^{0,\infty}.arr\_time$ (Línea 4). El *área de trabajo* de cada contacto en el CP debe ser limpiado antes de cada búsqueda de Dijkstra, previamente hecho en 3.20 (Línea 32) mediante 3.26. Un área de trabajo despejada significa: (i) tiempo de llegada del contacto $C.arr\_time = \infty$, (ii) indicador de contacto visitado $C.visited = False$, (iii) predecesor del contacto $C.pred = \varnothing$, y (iv) lista de nodos visitados $C.visited\_n \leftarrow \{\}$ (Todo esto preparado en la función anterior 3.20, líneas 14-25)

Inicialmente el contacto raíz "$root$" es asignado a "$current$" 3.21, línea 12).

Obsérvese que la lista de nodos visitados es una mejora de CGR que va más allá del algoritmo básico descrito en SABR [59].

Al principio se agrega el destino del contacto raíz en la lista de nodos visitados "$root->visitedNodes$" (Líneas 14-16).

Luego, dentro de un ciclo infinito (Línea 17), se explora el plan de contactos manteniendo los parámetros del contacto actual "$current$". Los contactos sucesivos "$nextHop$" del contacto actual son los contactos cuyas fuentes son iguales a los destinos del contacto actual (Línea 21). El siguiente procedimiento está representado por el algoritmo 2, que se verifica para cada contacto sucesor "$nextHop$", donde se verifican las condiciones de descalificación de contacto "$neighborList->contact$":

- El tiempo de contacto sea menor al tiempo de llegada (Línea 23).
- El contacto ya ha sido visitado o suprimido anteriormente. (Línea 26)
- El contacto continúa con otro que ya ha sido visitado (Línea 29).

A continuación se calcula el tiempo de llegada "$arrTime$" del mejor caso (costo) para la transmisión del paquete *Bundle* durante el contacto (Líneas 34-40), si la hora de llegada calculada es anterior a cualquier hora de llegada calculada previamente para el contacto "$neighborList- > contact$" (Línea 43), indica que la hora de llegada para el contacto se mejora transmitiendo desde el contacto actual (Líneas 45-46), entonces la nueva hora de llegada y el correspondiente nodo predecesor del contacto actual se anotan en el área de trabajo del contacto, como también los contactos visitados para la siguiente iteración de ciclo (Líneas 56-62).

Al identificar los contactos que llegan a destino se debe cumplir dos condiciones: que el destino del último contacto en la ruta sea el mismo que el destino del paquete "*Bundle*" y que no exista un tiempo de llegada menor al tiempo de llegada final (Línea 65), de ser así los nuevos parámetros se anotan para una posible siguiente iteración del ciclo.

Una vez realizados los pasos para cada contacto sucesor del contacto actual, se marca como "visitado"(Línea 79) y comienza el Procedimiento de Selección de Contactos.

Una vez revisado el plan de contactos y actualizadas las áreas de trabajo de sus contactos por el CRP [2], el procedimiento del CSP [3] vuelve a visitar todos los contactos del plan para determinar cuál es el que tiene la mejor hora de llegada anotada "$bestArrTime$" (Líneas 85-97).

Este conjunto de nodos se conoce como frontera en la terminología de Dijkstra [114]. Cada contacto de la frontera que ya haya sido visitado o que esté suprimido administrativamente se salta inmediatamente (Líneas 87-88). Cada contacto cuya hora de llegada sea posterior a la "$bestFinArrTime$" actual también se salta inmediatamente (Líneas 89-90). Es en este punto donde debe cumplirse la hipótesis de Dijkstra de una métrica de costes monotónicamente creciente. En efecto, dado que el tiempo no puede retroceder, no tiene sentido continuar con la exploración a través de dicho contacto.

De los contactos restantes, el que tenga la hora de llegada "$arrivalTime$" más temprana se asigna al contacto siguiente "$next$"(Línea 94).

Al salir del ciclo *for*, si no existe un contacto siguiente, termina el ciclo principal o de lo contrario el contacto siguiente se asigna a la variable del contacto actual y el ciclo vuelve a comenzar (Línea 99-100).

El contacto final "$finalContact$" es el contacto que recibe el paquete de datos, el nodo destino de la fuente. Si este contacto se alcanza, en el ciclo de exploración del plan de contactos, la mejor ruta queda almacenada en "$finalContact$" y se deben reconstruir las ruta.

El ciclo de reconstrucción de ruta se entiende de la siguiente manera:

La secuencia de contactos de la ruta se recupera a partir de los contactos predecesores "$ctc- > predecessor$", empezando por "ctc" y volviendo al contacto raíz "$root$", mientras se rellena la lista de saltos *hops* de la ruta resultante (líneas 107-110). Se reordena la ruta y se anotan los parámetros relevantes (Línea 112-118) de modo que el primer contacto sea la fuente, entonces el primer salto ("$firstHop$") será el primer nodo en la ruta.

La ruta se formó en el procedimiento de revisión de contactos (CRP) 2, y los parámetros de nodo destino y nodo siguiente son directamente accesibles a través del primer salto (Línea 127) y el último salto, que será el destino del contacto final.

Los últimos cálculos a realizarse son, el límite de volumen de ruta "$Route- > capacity$" y la cantidad de saltos que tiene la ruta "$Route- > hopCount$"(Líneas 126-138).

Por último se retorna la ruta seleccionada (Línea 149).

```
1 struct routeStr *dijkstra(Contact *root, node dest, Contact *plan) {
2
3   time_t bestArrTime, bestFinArrTime, arrTime;
```

```
4   time_t rtToTime=LONG_MAX;
5   Contact *finalContact, *current, *next, *ctc;
6   finalContact = current = next = ctc = NULL;
7   struct routeStr *Route = NULL;
8   struct contactList *neighborList,*nlist ;
9   neighborList = nlist = createNeighborList(plan);
10  nodeList *nl;
11  bestFinArrTime=LONG_MAX;          //tiempo de llegada final
12  current = root;
13
14  if (!isInNodeList(root->visitedNodes,root->destination)){
15    root->visitedNodes = addToNodeList(root->visitedNodes, root->destination);
16  }
17  while(TRUE){
18
19    for(neighborList=nlist;neighborList!=NULL;neighborList=neighborList->next){
20
21      if(current->destination == neighborList->contact->source){
22        //ignoro contactos que terminan antes que el tiempo de llegada
23        if(neighborList->contact->end <= current->arrivalTime)
24          continue;
25        //ignoro si el contacto ha sido visitado o suprimido
26        if(neighborList->contact->visited || neighborList->contact->suppressed)
27          continue;
28        // ignoro contactos que llevan a nodos visitados previamente
29        if(isInNodeList(current->visitedNodes,
30            neighborList->contact->destination))
31          continue;
32
33        //calculo el costo (arrival time)
34        if(neighborList->contact->start < current->arrivalTime){
35
36          arrTime = current->arrivalTime ;
37
38        }else
39          arrTime = neighborList->contact->start;
40
41        //actualizo el costo si hay uno mejor o igual
42
43        if(arrTime <= neighborList->contact->arrivalTime){
44
45          neighborList->contact->arrivalTime = arrTime;
46          neighborList->contact->predecessor = current;
47
48          nl = neighborList->contact->visitedNodes;
49          for( ;nl != NULL; ){
50            neighborList->contact->visitedNodes = nl;
51            nl=nl->next;
52            vPortFree(neighborList->contact->visitedNodes);
53          }
54          neighborList->contact->visitedNodes = NULL;
55
56          for(nodeList *nl = current->visitedNodes;nl!=NULL;nl=nl->next)
57            neighborList->contact->visitedNodes=
58              addToNodeList(neighborList->contact->visitedNodes, nl->nInL);
59
60          neighborList->contact->visitedNodes =
61            addToNodeList(neighborList->contact->visitedNodes,
62            neighborList->contact->destination);
```

```
63
64          // marcar si llega a destino
65          if (neighborList->contact->destination == dest
66            && neighborList->contact->arrivalTime < bestFinArrTime){
67            //actualizo el tiempo de llegada final
68            bestFinArrTime = neighborList->contact->arrivalTime;
69            finalContact = neighborList->contact;
70
71          }
72
73        }
74
75      }
76
77    }
78    //Revision del contacto completada
79    current->visited = TRUE;
80
81    //determinar el mejor proximo contacto en todo el plan
82    bestArrTime = LONG_MAX;
83    next = NULL;
84
85    for(ctc=plan;ctc!=NULL;ctc=ctc->next){
86
87      if(ctc->suppressed || ctc->visited)
88      continue;
89      if(ctc->arrivalTime > bestFinArrTime)
90      continue;
91      if(ctc->arrivalTime < bestArrTime)
92      {
93        bestArrTime = ctc->arrivalTime;
94        next = ctc;
95      }
96
97    }
98
99    if(next==NULL) break;
100   current = next;
101
102 }
103
104 if(finalContact != NULL){
105
106   ctc=finalContact;
107   while(ctc->predecessor != root && ctc->predecessor != NULL)
108   ctc=ctc->predecessor;
109   ctc->predecessor = NULL;
110   finalContact->sucessor = NULL;
111
112   for(current=finalContact;current!=NULL;current=current->predecessor){
113     if(current->predecessor != NULL) {
114       ctc=current->predecessor;
115       ctc->sucessor = current;
116     }
117     if(current->end < rtToTime) rtToTime = current->end;
118   }
119
120   //ahora anoto parametros relevantes de la ruta
121   Route = (struct routeStr *)pvPortMalloc(sizeof(struct routeStr));
```

```
122     if(Route == NULL){
123       printf("pvPortMalloc in Route dijkstra fail");
124       return -1;
125     }
126     Route->ctcList  = ctc;
127     Route->firstHop =ctc->destination;
128     Route->hopCount =0;
129     Route->toTime =rtToTime;
130     Route->arrivalTime= ctc->arrivalTime;
131     Route->capacity =  UINT_MAX;
132     for(current=ctc; current!=NULL;current=current->sucessor){
133       Route->hops = addToNodeList(Route->hops,current->destination);
134       Route->hopCount++;
135       if(current->capacity <= Route->capacity)
136         Route->capacity = current->capacity;
137     }
138     //printf("dijkstra Rt cap: %u\n", Route->capacity);
139   }
140   nlist = neighborList;
141   for(;nlist != NULL;){
142     neighborList = nlist;
143     nlist=nlist->next;
144     vPortFree(neighborList);
145 
146   }
147   //printRoute(Route);
148   return Route;
149 }
```

Código 3.21 – *Estructura Algoritmo de Dijkstra*

3.2.9. Estructura de Rutinas de Manejo de Listas Enlazadas

sta subrutina es utilizada en llamadas para almacenar nodos en una lista, como por ejemplo la lista de nodos visitados o suprimidos. Funciona recibiendo el puntero de la lista y el nodo que se desea agregar a esa lista (Línea 1). Luego, mediante asignación dinámica de memoria se asigna el nodo a una lista (Líneas 3-9), el nodo siguiente de la lista apunta (enlace) a la cabecera del nodo recibido (Línea 10) y se retorna la nueva cabecera de la lista (Línea 11).

```
1  nodeList *addToNodeList(nodeList *nl,  node nodo){
2  
3    nodeList *nw;
4    nw = pvPortMalloc(sizeof(nodeList));
5    if(nw == NULL){
6      printf("pvPortMalloc fail");
7  
8    }
9    nw->nInL = nodo;
10   nw->next=nl;
11   return nw;
12 }
```

Código 3.22 – *Creación de Nodos en CGR*

Para buscar datos en los nodos de la lista enlazada (Plan de Contactos), se recorre la lista recibida en un ciclo *for* (Línea 3) hasta encontrar un nodo que tenga la misma información que el nodo buscado (Línea 4). Si el dato se encuentra en la lista, se retorna un valor distinto a cero (Línea 7).

```
unsigned int isInNodeList(nodeList *n, node nd){

  for (nodeList *nl=n;nl!=NULL;nl=nl->next)
     if (nd == nl->nInL){
      return 1;
     }
  return 0;
 }
```

Código 3.23 – *Función para buscar nodos en CGR*

De la misma manera que la anterior, pero con diferentes tipos de datos, la búsqueda de próximos nodos se realiza recorriendo la lista en la que se desea buscar el dato (Línea 3). Si el dato se encuentra en la lista, se retorna un valor distinto a cero.

```

  unsigned int isInPrxNodeList(prxnodeList *n, node nd){
    for (prxnodeList *nl=n; nl!=NULL; nl=nl->next){
      if (nd == nl->prxmNode.node){
        return 1;
      }
    }
    return 0;
  }


```

Código 3.24 – *Busqueda en Nodos proximos*

Las siguientes rutinas están pensadas para impresión en consola de las rutas. Cuando se recibe la ruta(Línea 1), en un ciclo "*for*" se imprimen los datos de las rutas hasta que llegue al final (Líneas 5-6).

```
  void printRoute(struct routeStr *rt){
    Contact *p;
    printf("Desde   hasta range\n");
    //for(; p!=NULL; p=p->predecessor)
    for(p=rt->ctcList; p!=NULL; p=p->sucessor)
    printf("%u\t%u\t%ld-%ld\n",p->source,p->destination,p->start,p->end);

  }
  void printRoutes(struct routeList *routes){
    if (!routes)  return;
    struct routeList *rts;
    Contact *p;
    taskENTER_CRITICAL();
    printf("\n****Impresion de rutas****\n");
    printf("Desde   hasta range\n");

    for(rts=routes;rts!=NULL;rts= rts->next){

      for(p=rts->route->ctcList; p!=NULL; p=p->sucessor)
      printf("%u\t%u\t%ld-%ld\n",p->source,p->destination,p->start,p->end);

      if(rts->next) printf("next Route\n");
    }
    printf("*************************\n");
    taskEXIT_CRITICAL();
  }
```

Código 3.25 – *Impresion de rutas*

Las subrutinas de limpieza de trabajo se ejecutan cada vez que la subrutina principal se ejecuta, llamada por la rutina "firstDepleted()" 3.17, para liberar la memoria de las variables auxiliares (Línea 5-10) que utiliza el algoritmo.

```
1  void clearWorkAreas(Contact *cp) {
2
3    for ( ; cp!=NULL; cp=cp->next){
4
5      cp->arrivalTime = LONG_MAX;
6      cp->capacity  = 0;
7      cp->predecessor = NULL;
8      cp->visited   = FALSE;
9      cp->residualCapacity = cp->capacity;
10     cp->visitedNodes=NULL;
11
12   }
```

Código 3.26 – *Rutina de limpieza de variables de trabajo*

3.3. Carga del algoritmo en la NanoMind

La computadora A712D está equipada con una interfaz de diagnóstico que permite la interacción con la computadora a través de una pantalla de depuración similar a una consola simple. La interfaz consta de una conexión por protocolo RS232.

La conexión USB es capaz de alimentar el A712D, por lo que no se necesita una fuente de alimentación externa para comenzar a probar y cargar el software. La interfaz se ejecuta a "500000 baudios, 8N1" Emulación RS232 a través de USB usando un controlador de dispositivo FTDI, como se ve en la Figura 3.2 por lo que al usar un programa de terminal conectado al puerto COM virtual, la pantalla de inicio aparecerá unos segundos después de conectar la interfaz de diagnóstico a la computadora A712D.

"*GomSpace*" proporciona un compilador ARM personalizado para la computadora A712D, que incluye la biblioteca newlib c. En general, se puede usar cualquier compilador arm-none-eabi, pero todo el código de "*GomSpace*" se compila y prueba con la compilación específica del fabricante. Entonces, para evitar problemas relacionados con diferentes compiladores, se recomienda usar el compilador provisto, el cual al momento de este estudio es el arm-none-eabi-gcc 4.6.4.

El compilador se instala en una PC de escritorio con un sistema operativo Linux y programas adicionales como *Minicom* (Monitor Serial) y *OpenOCD* (Herramienta de soporte para depuración, del inglés *Open On-Chip Debugger*) para trabajar con los sistemas embebidos.

El paquete de software utiliza un systema de compilación basado en *Python* llamado *waf* que reemplazan los metodos de compilación más comunes como *./configure, make, install* simplemente por el comando *./waf configure build* que utiliza un esquema recursivo para leer y compilar todo el proyecto dentro de la carpeta. También proporciona varios códigos para la configuración de distintas placas de la misma marca, con sus diferentes opciones.

La carga del programa que se ejecutará en la *NanoMind* puede hacerse mediante dos métodos. La primera es cagar como imagen *-rom*, recomendada para cargarse como versión final del código desarrollado. La segunda es cargar como imagen RAM que sirve como método rápido para hacer pruebas antes de cargarla en la memoria FLASH. Este software estará ligado a la dirección *0x50000000*, reservada para la herramienta *ftp-program* por el puerto serial.

Fig. 3.5 – *En la imagen se puede ver la computadora de abordo (OBC NanoMind 712c) sobre la mesa de trabajo, conectada a la computadora donde se compila el código a ejecutar. Laboratorio de la Unidad de Formación Superior UFS-CONAE.*

Cuando la imagen del programa ha sido cargada a la *RAM* la OBC saltará a la dirección de inicio y ejecutará el programa.

Se utiliza un cliente FTP (*FTP-Client*) para cargar el programa por medio de la terminal y el protocolo CSP montado sobre el enlace *RS232*, ejecutando, en la carpeta del proyecto, el comando .
waf upload. Luego de esto, se puede acceder y ejecutar los programas cargados mediante la consola en el monitor serial *Minicom*, configurado a 500000 baudios 8N1.

3.4. Campaña Experimental

Para realizar las pruebas de código se necesitó de la construcción de un plan de contactos acorde a las necesidades de las pruebas. Esto implica tener una basta cantidad de información en la memoria y notar el efecto de la sobrecarga de cálculos en el procesador. Este plan de contactos se obtuvo mediante simulaciones realistas de una formación tipo *Walker*, teniendo como nodos: diez estaciones de tierra, cien puntos de contacto no supervisados en tierra, y doce satélites. Las locaciones para las estaciones terrenas son específicas, y aleatorias para los otros puntos de contacto.

Para detallar, el plan de contactos utilizado parte de un nodo central, MOC conectado indefinida o continuamente a otros nodos. Vale la pena mencionar que esto en un escenario real no suele suceder, las estaciones terrenas no necesitan correr este algoritmo en una computadora de recursos limitados como lo es la *NanoMind* OBC. Aún así, para este experimento se considera interesante estresar el tiempo de ejecución de la *NanoMind* OBC.

Los cálculos de acceso fueron hechos mediante el software *AGI's System's ToolKit*, disponible en CONAE y expresado en el formato ION, para que el algoritmo simplificado pueda leerlo en la *NanoMind* OBC, de la misma manera que en ION. El plan de contactos resultante, mostrado en la Figura 3.6, comprende 9000 contactos entre estos nodos, de los cuales se van cargando en intervalos de a 100, para medir así el tiempo de ejecución en referencia a la

```
 1 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:02:00.370 Src=[9] Dst=[112]
 2 a contact +0000000 +0000121 09 112 1
 3 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:42.022 Src=[14] Dst=[112]
 4 a contact +0000000 +0000343 14 112 1
 5 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:15.732 Src=[15] Dst=[112]
 6 a contact +0000000 +0000316 15 112 1
 7 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:01:10.387 Src=[18] Dst=[120]
 8 a contact +0000000 +0000071 18 120 1
 9 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:03.850 Src=[31] Dst=[114]
10 a contact +0000000 +0000304 31 114 1
11 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:19.682 Src=[34] Dst=[111]
12 a contact +0000000 +0000320 34 111 1
13 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:03:16.392 Src=[50] Dst=[119]
14 a contact +0000000 +0000197 50 119 1
15 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:16.559 Src=[77] Dst=[114]
16 a contact +0000000 +0000317 77 114 1
17 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:03:53.085 Src=[85] Dst=[119]
18 a contact +0000000 +0000234 85 119 1
19 #Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:00:23.507 Src=[90] Dst=[114]
20 a contact +0000000 +0000024 90 114 1
```

Fig. 3.6 – *Plan de contactos utilizado para las pruebas. Resultado de cálculos de acceso realizados mediante el software System Tool Kit, de AGI.*

cantidad de contactos cargados. De esta manera se supone que a mayor cantidad de contactos, aumentará la cantidad de rutas encontradas, por lo tanto también aumentará el tiempo de ejecución y la cantidad de memoria utilizada.

Las corridas del algoritmo, para tener una referencia, están preparadas para que empiecen en este nodo central (nodo 300), pasen por una estación terrena, de allí a un satélite (o más) y de allí nuevamente a otro punto de contacto en tierra como destino final.

Por lo tanto y como resumen, esta subrutina se escribió en lenguaje C, una implementación de la subrutina CGR de ION adaptada para correr en computadoras de vuelo para pequeños satélites como lo es la NanoMind, con un procesador ARM.

Se creó la tarea CGR, dinámicamente en FreeRTOS y esta tarea carga el plan de contactos en la memoria de trabajo.

Seguidamente se cargó el programa, se ejecutó el algoritmo en la *NanoMind* y se midió el tiempo de ejecución en cada corrida con una cantidad de contactos conocida.

En cada ejecución sucesiva de la subrutina, manualmente se aumentó la cantidad de contactos cargados del plan de contactos y el destino final o nodo final del supuesto paquete.

Luego se analizó la variación del tiempo de ejecución del algoritmo, también la cantidad de memoria utilizada versus la cantidad de contactos del plan de contactos cargados.

Se analizó el algoritmo en función a las subrutinas que más se ejecutan y las que tardan más tiempo en ejecutarse, según cada escenario, con herramientas de pruebas de *software* y *code profiling*, perfilador de código como: *GNU-Debugger* y *Gprof*.

CAPÍTULO 4

Resultados

El código ejecutado en la OBC *NanoMind* muestra mediante una terminal, primeramente la cantidad de nodos cargados, todas las rutas encontradas con esos nodos, nodos próximos y el siguiente salto del paquete y finalmente los parámetros buscados, que son el tiempo de ejecución, la cantidad de memoria utilizada y la cantidad de memoria liberada.

4.1. Resultados - Subrutina CGR

En la terminal se puede ver que al tener como origen el nodo 2 y destino el nodo 11:

```
CGR
Nodo excluido: 5            (Se excluye al nodo 5 de las posibles rutas)
Plan cargado                (Confirmación de que se cargó el plan)
Tamaño del plan: 268        (Cantidad de contactos cargados en el plan)
```

Primera ruta:

Fuente	Destino	Empieza	Termina
2	10	1509	1677
10	4	1984	2321
4	7	2320	2690
7	11	2951	3264

Siguiente ruta:

Fuente	Destino	Empieza	Termina
2	10	2207	2525
10	11	2220	2619

Siguiente ruta:

Fuente	Destino	Empieza	Termina
2	5	1595	2034
5	9	1859	2184
9	1	2140	2400
1	3	2097	2339
3	11	2142	2405

Siguiente ruta:

Fuente	Destino	Empieza	Termina
2	3	1245	1456
3	11	1212	1269

```
   Proximate nodes: 3, 10, (Se muestra los posibles siguientes nodos)
Next hop : 3,              (Siguiente nodo seleccionado)
```

Debido a que se excluye el nodo 5, de todas las rutas encontradas se muestran solo los próximos nodos disponibles para transmitir el dato.

4.2. Análisis de Resultados

Además de encontrar los próximos nodos a quienes pueden transmitir los paquetes, se analizó la cantidad de rutas encontradas en función a la cantidad de contactos agregados al plan. En la Tabla **??**.

Cantidad de Rutas Encontradas Mediante CGR				
Cantidad de Contactos Cargados	Desde Nodo 300 a Nodo 14	Desde Nodo 300 a Nodo 19	Desde Nodo 300 a Nodo 28	Desde Nodo 300 a Nodo 13
100	1	1	0	0
200	3	2	1	0
300	4	4	2	1
400	5	5	2	1
500	6	6	2	2
600	8	8	2	2
700	9	9	2	2
800	10	10	2	2
900	11	11	2	2
1000	13	12	2	2
1100	13	12	2	2
1200	13	12	2	2
1300	13	12	2	2
1400	13	12	2	2

Tabla 4.1 – *Cantidad de rutas encontradas por cantidad de contactos agregados, para los nodos 14, 19, 28 y 13*

Luego del la escritura del código en lenguaje C, se depuró y se analizó la correcta ejecución del algoritmo con la herramienta GDB (*GNU Project Debugger*) para garantizar que se den los mismos pasos que en la librería de ION.

Mediante *Gprof*, un software que analiza el perfil del código, se pudieron detectar los cuellos de botella en la ejecución del algoritmo mostrados en la tabla **??**.

4.2.1. Tiempo de Ejecución

Los resultados, como es de esperarse, indican que la cantidad de nodos en la red no afectan directamente al tiempo de ejecución, como sí lo hace la cantidad de contactos entre estos nodos, que al fin y al cabo es lo que genera o conecta las rutas para que los paquetes *bundles* lleguen a destino. Entonces la cantidad de nodos, por vasta que sea, no afectaría a al tiempo de ejecución si la cantidad de contactos entre estos nodos se mantenga. El algoritmo itera sobre todos los contactos del plan encontrando rutas, teniendo como subrutina principal el algoritmo de Dijkstra [114]. Existe la posibilidad de que las rutas tengan demasiados contactos entre los nodos, por lo que el tiempo de ejecución para encontrar esta ruta será mayor, como también ocupará mayor cantidad de memoria al ejecutarse el algoritmo. Esto puede llegar necesitar demasiados recursos de memoria y provocar errores de alojamiento dinámico, haciendo que la tarea termine.

Para explicar este comportamiento con más profundidad, se utiliza el software *Gprof*, para

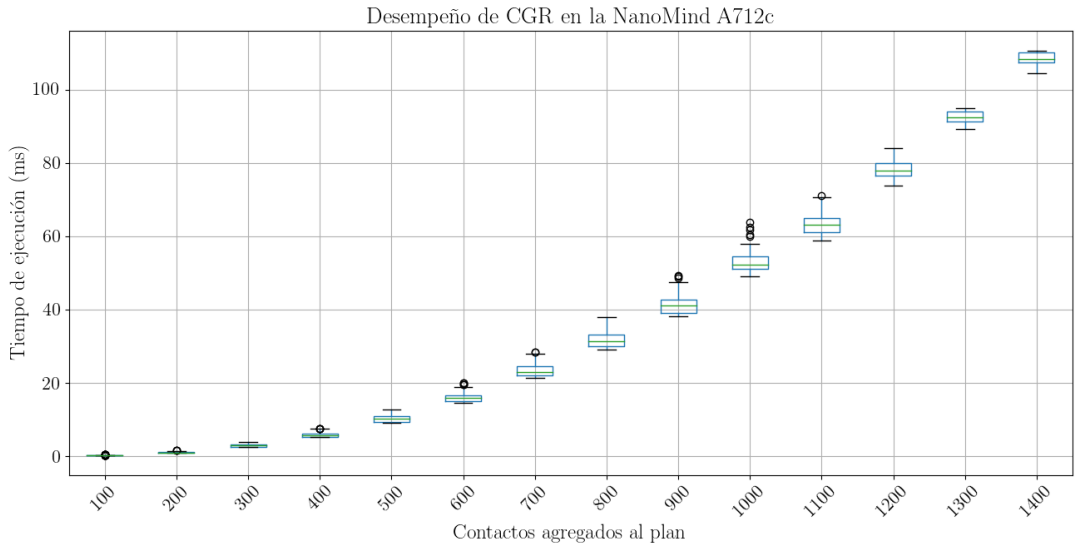


Fig. 4.1 – *Tiempo de ejecución de CGR para distintas cantidades de contactos agregados al plan.*

tener un perfil del código y así determinar que subrutinas consumen más tiempo o son las mas llamadas. Para este análisis se programó un *script* en lenguaje BASH que prepare el código con las banderas necesarias para este procedimiento. Dicho *script* se puede ver más abajo detalladamente.

Script de Análisis de Resultados en Bash

```
#!/bin/bash



# Recorro la cantidad de contactos de 100 en 100
for cContct in {100..1500..100}

  do
  # Primero recorro cada nodo, desde el nodo 11 hasta el 70
  for dst in {11..70}

    do

    #Luego ejecuto el programa principal con los flags para depurar
    taskset --cpu-list 1 ./cgr $dst $cContct

    #Con GPROF analizo el tiempo de cada subrutina del programa
    gprof -c cgrGprof gmon.out > File$(printf "%d-%d" "$dst" "$cContct").txt
    done

  done


```

Código 4.1 – *Código BASH para análisis de resultados*

De esta forma, se determinó que las subrutinas que más afectan al tiempo de ejecución son las que tratan directamente con la formación, búsqueda de parámetros, y modificación de datos en las listas enlazadas.

Llamadas a Funciones de CGR			
En menor cantidad de rutas encontradas			
A nodo 28, 1300 contactos		A nodo 13, 1300 contactos	
Cantidad de llamadas	Función	Cantidad de llamadas	Función
3	dijkstra	3	dijkstra
255031	addToNodeList	259187	addToNodeList
33881	isInNodeList	34409	isInNodeList
2024	clearWorkArea	2024	clearWorkArea
3	createNeighborList	3	createNeighborList
2	appendRoute	2	appendRoute
2	printRoute	2	printRoute
En mayor cantidad de rutas encontradas			
A nodo 14, 1000 contactos		A nodo 19, 1000 contactos	
14	createNeighborList	13	createNeighborList
14	dijkstra	13	dijkstra
254010	addToNodeList	254019	addToNodeList
33871	isInNodeList	33874	isInNodeList
2018	clearWorkArea	2018	clearWorkArea
13	appendRoute	12	appendRoute
13	printRoute	12	printRoute

Tabla 4.2 – *Cantidad de Llamadas a Cada Función*

Esta característica es debido al tipo de estructura que presenta el plan de contactos. Los contactos se alojan dinámicamente en la memoria (se realiza a través de funciones de la biblioteca estándar de C, malloc()), utilizando listas enlazadas para el manejo de datos. Prima la ventaja de poder actualizar el plan de contactos sin mucha complejidad (que puede aumentar o disminuir, según el resultados de propagadores orbitales), mediante una subrutina de lectura suficiente para cargar los datos a la memoria, y borrar los datos desactualizados. El almacenamiento permanente de los datos, en este caso, no es una opción.

Esta estructura de organizar los datos dentro del algoritmo, que se ajusta a la dinámica de CGR en el manejo de datos, implica que para acceder a un dato en específico, se tenga que recorrer la lista hasta encontrar el nodo que almacene el parámetro solicitado. Este recorrido también afecta al tiempo de ejecución del algoritmo.

4.2.2. Análisis de Memoria

También como es de esperarse, la cantidad de nodos, sí afecta a la cantidad de memoria utilizada, por lo que ésta es directamente proporcional al tamaño del plan de contactos, como se puede ver en la Figura 4.2.

Al iniciar el algoritmo, los datos de los contactos son cargados mediante asignación dinámica de memoria, por lo que se guardan en el *heap*. Entonces, a mayor cantidad de contactos, se cargan más datos en la memoria y por otro lado, existe mayor probabilidad de encontrar más rutas para llegar a destino. El hecho de encontrar más rutas, implica enlazar más datos, y da como resultado una mayor utilización de memoria.

Si los recursos de la memoria no son suficientes, durante la ejecución del programa, el sistema mostraba el mensaje:

Warning: Heap is running full trying to allocate 4096 bytes!! Heap start address = 0x50067250 Heap end address = 0x50150000 Current heap address = 0x50150000

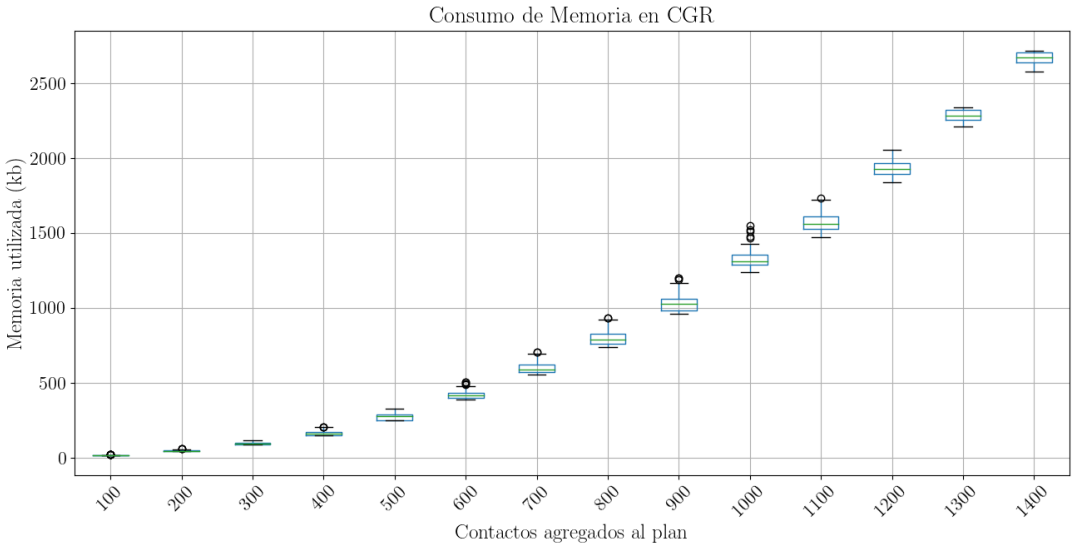


Fig. 4.2 – *Memoria utilizada durante la ejecución de CGR, para distintas cantidades de contactos agregados al plan*

Por lo tanto, cuando se encuentra mayor cantidad de rutas, mayor información se guarda en la memoria, entonces el programa solo funciona utilizando menor cantidad de contactos.

Para realizar el conteo de memoria asignada (en FreeRTOS) se utilizó la función propia del sistema operativo llamada "pvPortMalloc" modificando la librería con una variable global que se incrementa cada vez que esta función es llamada.

Esta función se utiliza para que la asignación dinámica de memoria se haga de forma segura, primeramente suspendiendo todas las tareas para utilizar la función malloc() y continuando nuevamente las tareas (Línea 5-11). Si no se ha conseguido asignar memoria, la función puede llamar a otra que gestione la asignación de memoria fallida. Luego de esto, el conteo de memoria aumenta en xWantedSize (Línea 27) para registrar la cantidad de memoria utilizada.

```
1   void *pvPortMalloc( size_t xWantedSize )
2   {
3     void *pvReturn;
4     /* El planificador suspende todas las tareas */
5     vTaskSuspendAll();
6     {
7       /* Se asigna memoria */
8       pvReturn = malloc( xWantedSize );
9     }
10    /* Se vuelven a tomar las tareas desde donde se haya suspendido */
11    xTaskResumeAll();
12
13    /* Si existe una funcion que utiliza este flag del sistema operativo*/
14    #if( configUSE_MALLOC_FAILED_HOOK == 1 )
15    {
16      if( pvReturn == NULL )
17      {
18        extern void vApplicationMallocFailedHook( void );
19
20        /* Se llama a la funcion de manejo de asignacion fallida */
21        vApplicationMallocFailedHook();
22      }
23    }
24    #endif
25
26    /* Aqui se almacena la cantidad de Memoria Usada */
27    memCount(xWantedSize);
28
29    return pvReturn;
30  }
31
32
```

Código 4.2 – *Conteo de memoria asignada en FreeRTOS para heap 3*

De la misma manera, la función "vPortFree" realiza la liberación de la memoria previamente asignada, suspendiendo las tareas para continuarlas luego de la liberación. Al final, la variable memFreedCount(sizeof (pv)) registra cuanta memoria es liberada.

```
1  void vPortFree( void *pv )
2  {
3    if( pv )
4    {
5      /* Se suspenden todas las tareas */
6      vTaskSuspendAll();
7      {
8        /* Se libera la memoria utilizada*/
9        free( pv );
10     }
11     /* Las tareas continuan corriendo */
12     xTaskResumeAll();
13
14     /* Conteo de memoria liberada */
15     memFreedCount(sizeof(pv));
16   }
17 }
18
19
```

Código 4.3 – *Conteo de memoria liberada en FreeRTOS para heap 3*

Estas funciones son llamadas en la tarea CGR para todos los experimentos realizados.

```
void cgr (void *pvParameters) {

  /* Variables a utilizar */
  Contact *cp, *cpaux;  //cabeceras del plan de contactos (listas enlazadas)
  Bundle *bundleP;     //Supuesto dato a enviar
  nodeList *En=NULL;     //Lista de nodos/rutas

/* creo el supuesto dato con todos sus parametros. Comienza en el nodo 300
 * y el nodo receptor de los datos es el 22
 */
  bundleP=create_bundle(300, 22);

/* agrego los nodos que debe excluir */
  node exNode = 5;
  En = addToNodeList(En, exNode);

/* configuro la cantidad de contactos
 * que se carga en la memoria
 */
  int i =1500;

/* Contadores de memoria a cero */
  TotalMem = 0;
  freedMem = 0;
  printf("En %d nodos:",i);

/* cargo el plan */
  cp=load_plan(i);  //printf("Plan cargado\n");

/* Comienzo a contar antes de ejutar la subrutina cgr */
  portTickType start = xTaskGetTickCount();

/* start CGR */
  cgrForward(bundleP, cp, En);

/* Termino de contar al retornar de la subrutina cgr */
  portTickType stop = xTaskGetTickCount();

/*  Imprimo tiempo de ejecucion
 *      memoria usada
 *      memoria liberada
 */
  printf("Ex time:\t%lu\t
          memU:%d\t
        memlib:%d\n"
        ,stop-start,
        totalMem,
        freedMem);
  fflush(stdin);
}

```

Código 4.4 – *Tarea CGR creada en FreeRTOS*

[46]

Para medir la eficiencia en la gestión de recursos, se analiza la diferencia entre la memoria utilizada y la memoria liberada $MemoriaUtilizada - MemoriaLiberada$. El resultado se puede ver en la Figura 4.3

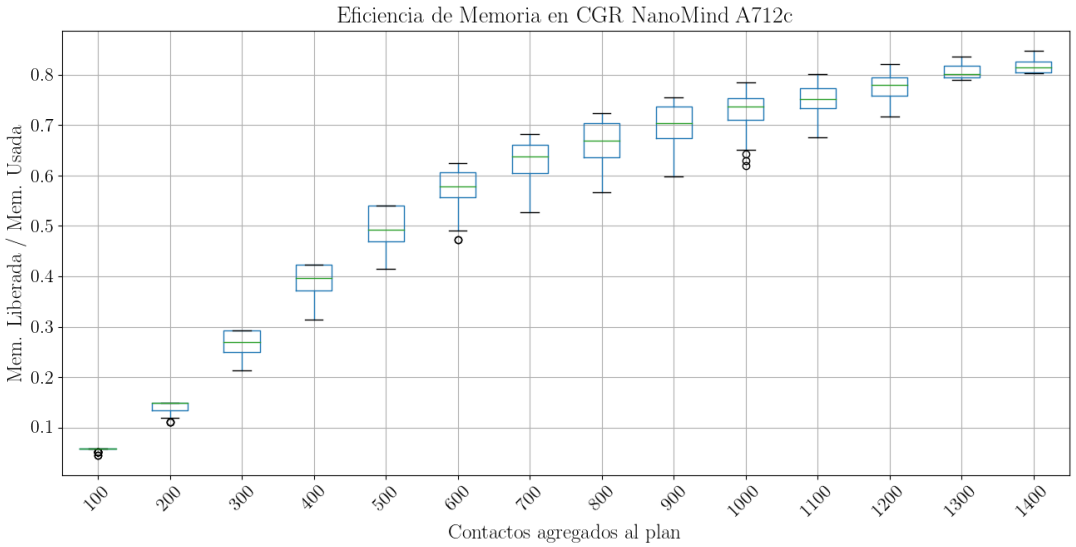


Fig. 4.3 – *Eficiencia de uso de memoria*

4.2.3. Ajuste de Curvas de los Resultados

Para completar el análisis, se toma la mediana de los datos presentados en la Tabla **??** y se busca una ecuación, mediante regresión lineal, que pueda sugerir valores para N cantidad de contactos. Con ayuda del lenguaje *Python* se utilizó la librería *Numpy* y la función *numpy.polifit()* para encontrar curvas que representen el fenómeno y posteriormente elegír cual de estas curvas encaja mejor con los datos obtenidos en el experimento.

El valor Adj_R^2 es una versión modificada de R^2 que se ajusta por cada número de variables predictoras en un modelo de regresión lineal.

Se calcula de la siguiente manera:

$$Adj\_R^2 = 1 - \frac{(1 - R^2) * (n - 1)}{(n - k - 1)} \tag{4.1}$$

donde:

R^2: Es el valor R^2 del modelo. n: Es el número de observaciones. k: Es la cantidad de variables predictoras.

Primero se buscaron las medianas de los tiempos de ejecución t para cada cantidad de contactos Nc.

Con ese set de datos y con ayuda de *Python* se procede a la búsqueda de curvas con cinco modelos experimentales, donde el número indica la cantidad de coeficientes del polinomio.

Mediana de Valores de Tiempos de Ejecución														
Nc:	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400
t:	0.266	0.941	2.8135	5.558	10.2855	15.8255	22.9135	31.246	41.111	52.332	63.026	77.813	92.405	108.449

Tabla 4.3 – *Mediana de Tiempos de Ejecución (en segundos) encontradas por cantidad de contactos agregados al plan]*

```
1
2 #Funcion para encontrar el valor r-cuadrado ajustado.
3 def adjR (x, y, degree) :
4 results = {}
5 coeffs = np.polyfit(x, y, degree )
6 p = np.poly1d(coeffs)
```

```
yhat = p(x)
ybar = np.sum(y)/len(y)
ssreg = np.sum((yhat-ybar) **2)
sstot = np.sum((y-ybar) **2)
results['r_squared'] = 1-(((1 -(ssreg/sstot))*(len(y)-1))/(len(y)-degree-1))

return results

import pandas as pd
import numpy as np

#create DataFrame Ex_ime
df = pd.DataFrame({

  'x': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120,
  130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240,
  250, 260, 270, 280, 290, 300, 400, 500, 600, 700, 800, 900,
  1000, 1100, 1200, 1300, 1400],

  'y': [0.029, 0.045, 0.063, 0.08, 0.102, 0.128, 0.164, 0.203, 0.228,
  0.266, 0.285, 0.35, 0.382, 0.458, 0.528, 0.572, 0.656, 0.748, 0.844,
  0.941, 1.112, 1.208, 1.332, 1.47, 1.606, 1.755, 2.212, 2.375,
  2.579, 2.8135, 5.558, 10.2855, 15.8255, 22.9135, 31.246,
  41.111, 52.332, 63.026, 77.813, 92.405, 108.449]})

#ajuste de modelos polinomiales hasta grado 5
model1 = np.poly1d (np.polyfit(df.x, df.y, 1))
model2 = np.poly1d (np.polyfit(df.x, df.y, 2))
model3 = np.poly1d (np.polyfit(df.x, df.y, 3))
model4 = np.poly1d (np.polyfit(df.x, df.y, 4))
model5 = np.poly1d (np.polyfit(df.x, df.y, 5))

#Valores R-squared calculados para cada modelo
print (adjR(df.x, df.y, 1))
print (adjR(df.x, df.y, 2))
print (adjR(df.x, df.y, 3))
print (adjR(df.x, df.y, 4))
print (adjR(df.x, df.y, 5))
```

Código 4.5 – *Script en Python para encontrar el polinomio mejor ajustado a los resultados optenidos*

Que da como resultado los valores R^2:
Modelo 1: 'rsquared': 0,9014705767538977,
Modelo 2: 'rsquared': 0,9998729386180556,
Modelo 3: 'rsquared': 0,9998745995712747,
Modelo 4: 'rsquared': 0,999869157985078,
Modelo 5: 'rsquared': 0,9998721812457333

Estos valores nos indica el porcentaje de la variación de la variable de respuesta que puede explicarse por la(s) variable(s) predictora(s) del modelo, ajustado por el número de variables predictoras. Como se puede ver, el valor '*rsquared*' que más se acerca a 1 es el modelo 3 (Línea 35), cuya curva se puede ver en la Figura 4.4.

Entonces la ecuación que representa el desempeño en referencia a tiempo de ejecución del algoritmo CGR es:

$$T_E = -1,859e^{-9}x^3 + 6,035e^{-5}x^2 - 0,01093x + 0,4558 \tag{4.2}$$

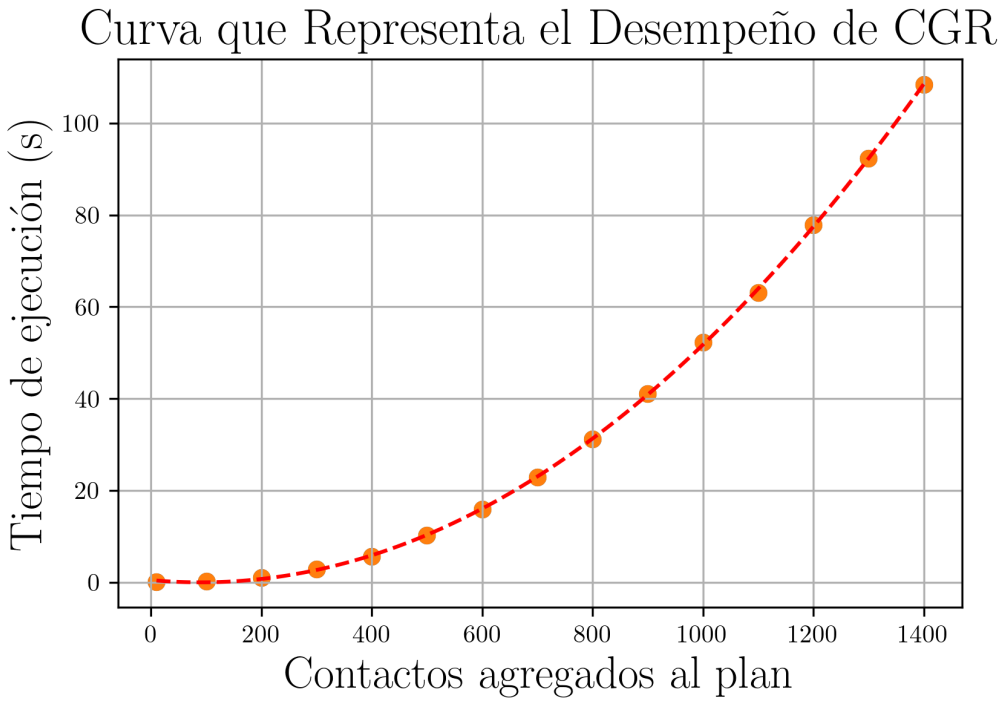


Fig. 4.4 – *Curva que representa el desempeño del algoritmo CGR corriendo en la OBC NanoMind.*

En cuanto a la cantidad de memoria utilizada, se sigue el mismo procedimiento: Tomar os datos del experimento y usar el *script* anterior para tener la representación de la Memoria Utilizada por el algoritmo durante su ejecución.

En este caso el resultado los valores R^2 son :
Modelo 1: 'rsquared': 0.9056720992990083,
Modelo 2: 'rsquared': 0.999769021047678,
Modelo 3: 'rsquared': 0.999799618108212,
Modelo 4: 'rsquared': 0.9998036045787224,
Modelo 5: 'rsquared': 0.9998006705168712

Por lo tanto, el valor más representativo es el modelo 4, polinomio de grado 4, cuya curva se puede ver en 4.5.

Y la ecuación de describe la curva es:

$$M_U = -1{,}623e^{-7}x^4 + 0{,}0005478x^3 + 0{,}9401x^2 - 42{,}66x + 8599 \tag{4.3}$$

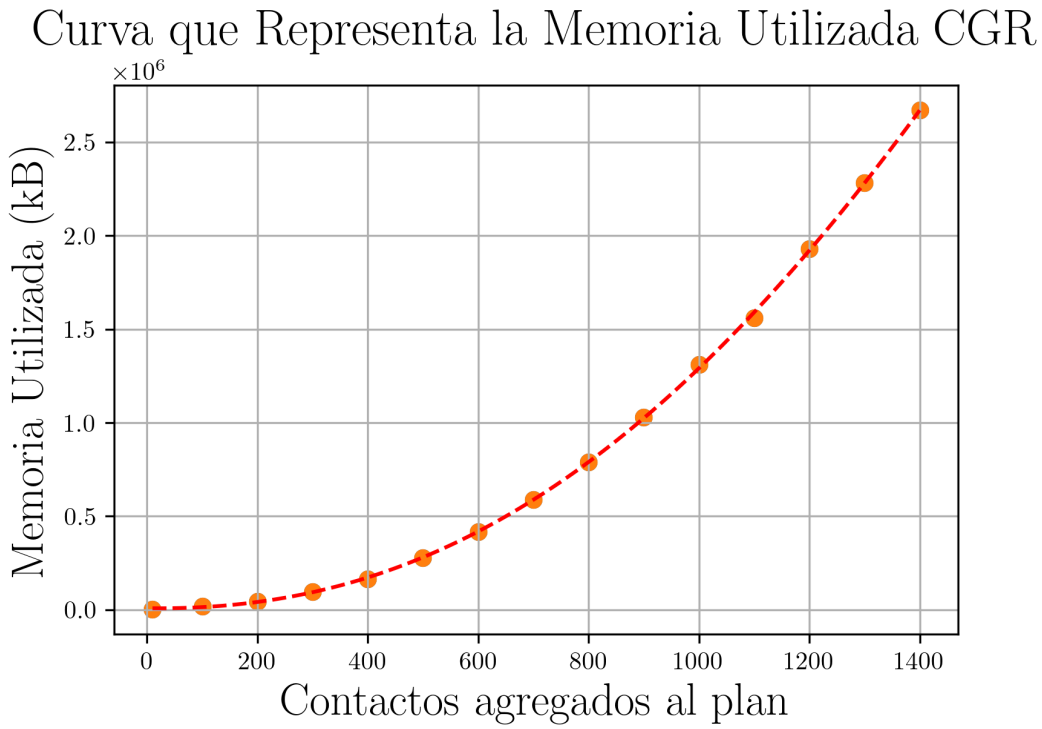


Fig. 4.5 – *Curva que representa la Memoria Utilizada por el algoritmo CGR corriendo en la OBC NanoMind.*

Conclusiones

En este trabajo se evaluó el rendimiento del algoritmo CGR implementado para correr en una computadora de capacidad limitada, como lo es la *NanoMind* A712c. Para ello se re-implementaron algoritmos de un código complejo, como lo es ION, pensada para correr en naves espaciales con computadoras de mayor poder de procesamiento. Se explicó en detalle la nueva implementación, y la adaptación de esta como tarea dentro de un sistema operativo. Se habló también de los detalles de la computadora que corrió el código y se extrajeron datos de parámetros reales para varios escenarios de comunicación dentro de una red.

Se presentó el estado del arte de los últimos avances sobre constelaciones satelitales y misiones que se benefician de sus características, incluyendo las primeras misiones que sentaron las bases para la actualidad, donde se ve el éxito de su implementación. La utilización de un sistema operativo que permita abstraerse de la coordinación de subrutinas para cada tarea a efectuar en una OBC es recomendada, bastante probada y documentada. Soluciones como FreeRTOS son utilizadas por fabricantes de OBC comerciales y se adquieren con subrutinas de control de dispositivos desarrollados para facilitar la implementación de las misiones satelitales con pequeños satélites. Permiten el desarrollo de software en lenguaje C/C++, bastante utilizado para programación de microprocesadores. En el trabajo se implementó el algoritmo CGR en lenguaje C por su eficiencia y cercanía al lenguaje de máquina, obteniéndose resultados valiosos que sirven como base de estudio para utilizar el protocolo en constelaciones de pequeños satélites. Sin embargo los resultados de memoria utilizada están sujetas a situaciones no determinísticas al trabajar con asignación dinámica de memoria y la utilización de implementaciones de malloc() basadas en "heap" que generalmente inducen la fragmentación de la memoria. v Se evidencia la eficiencia y utilidad del algoritmo de Dijkstra, mediante el cual se calculan todas las rutas disponibles para transmitir un dato y mediante otros algoritmos se obtiene un listado de nodos vecinos capaces de enviar el paquete a destino. La implementación se realizó como tarea de un sistema operativo de código abierto y se publicaron los resultados preliminares de la implementación en [3] y el código fuente en la página web: `https://github.com/vemol/CGR_NanoMind_A712c/find/master` Por último, se verificaron parámetros más relevantes del algoritmo implementado, utilizando herramientas de diagnóstico de software conocidas y de código abierto, para dar servicio a constelaciones satelitales que necesiten innovadores sistemas de comunicación en redes que toleren el retraso.

Trabajos futuros

El desarrollo de una red intersatelital/interconstelación es posible si se atacan los principales inconvenientes de comunicación. Es importante establecer bases para la resolución de asuntos como optimizar la propagación de la demora o la confiabilidad en la llegada de datos a destino. Este trabajo se podrá usar como aproximación, o modelo, para tomar decisiones en cuanto a costos de comunicación en una red, y estudiar la propagación de los costos a medida que la red aumenta en nodos. Al compartir los resultados de este trabajo, se aporta a la democratización del espacio debido a que un estudio similar, de una implementación más básica de CGR, y en lenguaje de bajo nivel, no es conocida por el autor. Esto implica que el código se puede portar a innovadoras misiones de pequeños satélites, por cualquier agencia o institución en el mundo. Otro tema importante a estudiar a futuro son las arquitecturas en las que los nodos pueden descargar sus datos indirectamente (por ejemplo, a través de redes ISL) también podrían reducir la latencia del acceso a los datos y lograr la vigilancia casi en tiempo real de zonas objetivo extensas. Al mismo tiempo, el aumento del número de satélites de observación también puede lograr una cobertura más amplia con altas resoluciones sin necesidad de franjas más amplias. Las tareas de planificación de misión implican la selección de componentes a medida, por lo que es pertinente la definición de criterios específicos para cada misión. La medición de parámetros de ejecución de software proporciona información valiosa para toma de decisiones, por lo que el autor propone realizar los mismos procedimientos pero con mayor cantidad de tareas para aumentar la confiabilidad del software de vuelo.

Otra opción es estudiar la implementación de sistemas con asignación de memoria estática, incluso sin el soporte de FreeRTOS para varios asignadores de memoria, implementar un asignador determinístico de bloques fijos de memoria de forma que sea fácilmente portable a cualquier RTOS. Basta con pre-asignar un pool (o *pools*) de bloques de memoria (ya sea estáticamente o desde el *heap* estándar antes de que comience la programación en tiempo real), y rellenar una cola RTOS con punteros a los bloques. La asignación puede realizarse simplemente tomando un puntero de la cola, y la desasignación devolviendo el puntero a la cola. Esto puede estudiarse en misiones de baja complejidad o donde los requisitos de cantidad de datos de trabajo sea determinística y por lo tanto pueda conocerse con anterioridad más parámetros de software. Para finalizar, queda como propuesta el estudio de este código mientras se ejecuta en conjunto con otros algoritmos de control utilizados en una misión real y modificando el algoritmo de Dijkstra para que también se considere la capacidad de almacenamiento en la selección de rutas.

ANEXOS

ANEXO A

Plan de Contactos Utilizado

#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:02:00.370 Src=[9] Dst=[112]
a contact +0000000 +0000121 09 112 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:42.022 Src=[14] Dst=[112]
a contact +0000000 +0000343 14 112 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:15.732 Src=[15] Dst=[112]
a contact +0000000 +0000316 15 112 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:01:10.387 Src=[18] Dst=[120]
a contact +0000000 +0000071 18 120 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:03.850 Src=[31] Dst=[114]
a contact +0000000 +0000304 31 114 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:19.682 Src=[34] Dst=[111]
a contact +0000000 +0000320 34 111 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:03:16.392 Src=[50] Dst=[119]
a contact +0000000 +0000197 50 119 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:16.559 Src=[77] Dst=[114]
a contact +0000000 +0000317 77 114 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:03:53.085 Src=[85] Dst=[119]
a contact +0000000 +0000234 85 119 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:00:23.507 Src=[90] Dst=[114]
a contact +0000000 +0000024 90 114 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:19.687 Src=[111] Dst=[34]
a contact +0000000 +0000320 111 34 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:42.027 Src=[112] Dst=[14]
a contact +0000000 +0000343 112 14 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:02:00.375 Src=[112] Dst=[9]
a contact +0000000 +0000121 112 09 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:15.736 Src=[112] Dst=[15]
a contact +0000000 +0000316 112 15 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:16.563 Src=[114] Dst=[77]
a contact +0000000 +0000317 114 77 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:00:23.511 Src=[114] Dst=[90]
a contact +0000000 +0000024 114 90 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:05:03.855 Src=[114] Dst=[31]
a contact +0000000 +0000304 114 31 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:03:16.397 Src=[119] Dst=[50]
a contact +0000000 +0000197 119 50 1
#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:03:53.090 Src=[119] Dst=[85]
a contact +0000000 +0000234 119 85 1

#Start=1 Jul 2017 00:00:00.000 Stop=1 Jul 2017 00:01:10.391 Src=[120] Dst=[18]
a contact +0000000 +0000071 120 18 1
#Start=1 Jul 2017 00:00:05.986 Stop=1 Jul 2017 00:06:45.234 Src=[1] Dst=[121]
a contact +0000006 +0000406 01 121 1
#Start=1 Jul 2017 00:00:05.991 Stop=1 Jul 2017 00:06:45.239 Src=[121] Dst=[1]
a contact +0000006 +0000406 121 01 1
#Start=1 Jul 2017 00:00:19.131 Stop=1 Jul 2017 00:05:58.227 Src=[62] Dst=[121]
a contact +0000020 +0000359 62 121 1
#Start=1 Jul 2017 00:00:19.137 Stop=1 Jul 2017 00:05:58.231 Src=[121] Dst=[62]
a contact +0000020 +0000359 121 62 1
#Start=1 Jul 2017 00:00:49.677 Stop=1 Jul 2017 00:03:11.163 Src=[89] Dst=[114]
a contact +0000050 +0000192 89 114 1
#Start=1 Jul 2017 00:00:49.682 Stop=1 Jul 2017 00:03:11.168 Src=[114] Dst=[89]
a contact +0000050 +0000192 114 89 1
#Start=1 Jul 2017 00:01:21.393 Stop=1 Jul 2017 00:07:18.792 Src=[69] Dst=[114]
a contact +0000082 +0000439 69 114 1
#Start=1 Jul 2017 00:01:21.398 Stop=1 Jul 2017 00:07:18.797 Src=[114] Dst=[69]
a contact +0000082 +0000439 114 69 1
#Start=1 Jul 2017 00:02:03.115 Stop=1 Jul 2017 00:07:01.875 Src=[19] Dst=[112]
a contact +0000124 +0000422 19 112 1
#Start=1 Jul 2017 00:02:03.119 Stop=1 Jul 2017 00:07:01.881 Src=[112] Dst=[19]
a contact +0000124 +0000422 112 19 1
#Start=1 Jul 2017 00:02:10.934 Stop=1 Jul 2017 00:05:37.947 Src=[92] Dst=[116]
a contact +0000131 +0000338 92 116 1
#Start=1 Jul 2017 00:02:10.939 Stop=1 Jul 2017 00:05:37.952 Src=[116] Dst=[92]
a contact +0000131 +0000338 116 92 1
#Start=1 Jul 2017 00:02:12.536 Stop=1 Jul 2017 00:04:24.862 Src=[76] Dst=[116]
a contact +0000133 +0000265 76 116 1
#Start=1 Jul 2017 00:02:12.540 Stop=1 Jul 2017 00:04:24.868 Src=[116] Dst=[76]
a contact +0000133 +0000265 116 76 1
#Start=1 Jul 2017 00:02:13.950 Stop=1 Jul 2017 00:08:26.122 Src=[80] Dst=[111]
a contact +0000134 +0000507 80 111 1
#Start=1 Jul 2017 00:02:13.955 Stop=1 Jul 2017 00:08:26.127 Src=[111] Dst=[80]
a contact +0000134 +0000507 111 80 1
#Start=1 Jul 2017 00:03:04.184 Stop=1 Jul 2017 00:08:40.550 Src=[86] Dst=[120]
a contact +0000185 +0000521 86 120 1
#Start=1 Jul 2017 00:03:04.189 Stop=1 Jul 2017 00:08:40.560 Src=[120] Dst=[86]
a contact +0000185 +0000521 120 86 1
#Start=1 Jul 2017 00:03:09.533 Stop=1 Jul 2017 00:05:56.000 Src=[6] Dst=[112]
a contact +0000190 +0000356 06 112 1
#Start=1 Jul 2017 00:03:09.539 Stop=1 Jul 2017 00:05:56.005 Src=[112] Dst=[6]
a contact +0000190 +0000357 112 06 1
#Start=1 Jul 2017 00:03:13.930 Stop=1 Jul 2017 00:07:17.236 Src=[32] Dst=[119]
a contact +0000194 +0000438 32 119 1
#Start=1 Jul 2017 00:03:13.935 Stop=1 Jul 2017 00:07:17.241 Src=[119] Dst=[32]
a contact +0000194 +0000438 119 32 1
#Start=1 Jul 2017 00:03:37.910 Stop=1 Jul 2017 00:09:48.364 Src=[10] Dst=[116]

a contact +0000218 +0000589 10 116 1
#Start=1 Jul 2017 00:03:37.914 Stop=1 Jul 2017 00:09:48.369 Src=[116] Dst=[10]
a contact +0000218 +0000589 116 10 1
#Start=1 Jul 2017 00:04:27.180 Stop=1 Jul 2017 00:10:41.393 Src=[30] Dst=[120]
a contact +0000268 +0000642 30 120 1
#Start=1 Jul 2017 00:04:27.185 Stop=1 Jul 2017 00:10:41.398 Src=[120] Dst=[30]
a contact +0000268 +0000642 120 30 1
#Start=1 Jul 2017 00:04:33.343 Stop=1 Jul 2017 00:10:10.423 Src=[90] Dst=[121]
a contact +0000274 +0000611 90 121 1
#Start=1 Jul 2017 00:04:33.348 Stop=1 Jul 2017 00:10:10.427 Src=[121] Dst=[90]
a contact +0000274 +0000611 121 90 1
#Start=1 Jul 2017 00:04:39.055 Stop=1 Jul 2017 00:10:09.926 Src=[61] Dst=[112]
a contact +0000280 +0000610 61 112 1
#Start=1 Jul 2017 00:04:39.060 Stop=1 Jul 2017 00:10:09.931 Src=[112] Dst=[61]
a contact +0000280 +0000610 112 61 1
#Start=1 Jul 2017 00:04:45.922 Stop=1 Jul 2017 00:10:20.393 Src=[46] Dst=[120]
a contact +0000286 +0000621 46 120 1
#Start=1 Jul 2017 00:04:45.928 Stop=1 Jul 2017 00:10:20.397 Src=[120] Dst=[46]
a contact +0000286 +0000621 120 46 1
#Start=1 Jul 2017 00:04:50.120 Stop=1 Jul 2017 00:10:36.676 Src=[47] Dst=[120]
a contact +0000291 +0000637 47 120 1

Referencias bibliográficas

[1] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, (New York, NY, USA), p. 27–34, Association for Computing Machinery, 2003.

[2] S. Burleigh, "Dynamic routing for delay-tolerant networking in space flight operations," in *SpaceOps 2008 Conference*, p. 3406, 2008.

[3] B. Vega and J. Fraire, "Experimental evaluation of on-board contact graph routing solutions for future nano-satellite constellations," 07 2020.

[4] S. Burleigh, T. De Cola, S. Morosi, S. Jayousi, E. Cianca, and C. Fuchs, "From connectivity to advanced internet services: A comprehensive review of small satellites communications and networks," *Wireless Communications and Mobile Computing*, vol. 2019, pp. 1–17, 05 2019.

[5] Z. Q. Zongfeng Ma, Lichao Sun, "The development of spacecraft electronic system," in *Communications in Computer and Information Science. Space Information Networks* (Q. Yu, ed.), vol. 688, (Kunming, China), Springer, August 2016.

[6] S. E. Inc., "Nano/microsatellite market forecast, 8 edition," 2018.

[7] CCSDS, *Red Book Schedule-Aware Bundle Routing.* CCSDS Secretariat, jul 2018. Draft Recommendation for Space Data System Standards. Draft Recommended Standard CCSDS 734.3-R-1.

[8] M. D. Graziano, "Overview of distributed missions," in *D'Errico M. (eds) Distributed Space Missions for Earth System Monitoring. Space Technology Library* (Springer, ed.), vol. 31, (New York, NY), pp. 375–386, 2013.

[9] S. Nag, "Satellite constellation mission design using model-based systems engineering and observing system simulation experiments," in *Proceedings of the Small Satellite Conference, Technical Session VIII: Frank J. Redd Student Competition, SSC14-VIII-1, https://digitalcommons.usu.edu/smallsat/2014/FJRStudentComp/1/*, 2014.

[10] S. Nag, J. LeMoigne, D. W. Miller, and O. de Weck, "A framework for orbital performance evaluation in distributed space missions for earth observation," in *2015 IEEE Aerospace Conference*, pp. 1–20, March 2015.

[11] S. Nag, T. Hewagama, G. T. Georgiev, B. Pasquale, S. Aslam, and C. K. Gatebe, "Multispectral snapshot imagers onboard small satellite formations for multi-angular remote sensing," *IEEE Sensors Journal*, vol. 17, pp. 5252–5268, Aug 2017.

[12] C. Araguz, E. Bou-Balust, and E. Alarcón, "Applying autonomy to distributed satellite systems: Trends, challenges, and future prospects," *Systems Engineering*, vol. 21, 03 2018.

[13] C. Iacopino and P. Palmer, "Autonomy," in *D'Errico M. (eds) Distributed Space Missions for Earth System Monitoring. Space Technology Library*, pp. 309–329, Microcosm Press and Springer, 08 2013.

[14] S. M. Schilling K., "Communication in distributed satellite systems," in *D'Errico M. (eds) Distributed Space Missions for Earth System Monitoring. Space Technology Library* (Springer, ed.), vol. 31, (New York, NY), pp. 345–354, 2013.

[15] J. A. Fraire and J. M. Finochietto, "Design challenges in contact plans for disruption-tolerant satellite networks," *IEEE Communications Magazine*, vol. 53, pp. 163–169, May 2015.

[16] CCSDS, *Bundle Protocol Specification.* Washington, DC, USA: CCSDS Secretariat, sep 2015.

[17] M. Langer and J. Bouwmeester, "Reliability of cubesats – statistical data, developers' beliefs and the way forward," in *Proceedings of the AIAA/USU Conference on Small Satellites, SSC16-X-2*, 08 2016.

[18] E. Kulu, "World's largest database of nanosatellites, over 3000 nanosats and cubesats." `https://www.nanosats.eu/`, 2020. [Online; accessed apr 2020].

[19] Union of Concerned Scientists, "Ucs satellite database." `https://www.ucsusa.org/resources/satellite-database`, 2020. [Online; accessed apr 2020].

[20] G. Tyc, L. Gomes, A. Cawthorne, A. da Silva Curiel, N. Navarthinam, and M. Sweeting, "Rich data, cheap satellites," in *Proceedings Of The AIAA/USU Conference On Small Satellites, Session 11: Big Data From Small Satellites 2, SSC17-XI-06,https://digitalcommons.usu.edu/smallsat/2017/all2017/151/*, 2017.

[21] M. L'Abbate, P. Venditti, C. Svara, F. Bagaglini, and R. Roscigno, "From mbps to gbps: Evolution of payload data handling and transmission system for future earth observation missions," in *2014 IEEE Metrology for Aerospace (MetroAeroSpace)*, pp. 576–581, May 2014.

[22] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, may 2008.

[23] B. D. Tapley, S. Bettadpur, M. Watkins, and C. Reigber, "The gravity recovery and climate experiment: Mission overview and early results," *Geophysical Research Letters*, vol. 31, no. 9, 2004.

[24] E. Gill, S. D'Amico, and O. Montenbruck, "Autonomous formation flying for the prisma mission," *AIAA Journal of Spacecraft and Rockets*, vol. 44, 05 2007.

[25] M. Zink, H. Fiedler, I. Hajnsek, G. Krieger, A. Moreira, and M. Werner, "The tandem-x mission concept," *2006 IEEE International Symposium on Geoscience and Remote Sensing*, pp. 1938–1941, 2006.

[26] M. Rodriguez-Cassola, S. V. Baumgartner, G. Krieger, and A. Moreira, "Bistatic terrasar-x/f-sar spaceborne–airborne sar experiment: Description, data processing, and

results," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, pp. 781–794, Feb 2010.

[27] P. P. Iacopino C., "Autonomy," in *Distributed Space Missions for Earth System Monitoring. Space Technology Library* (D. M., ed.), vol. 31, (New York, NY), Springer, 2013.

[28] L. J. Ippolito, *Satellite Communications Systems Engineering.* The Atrium, Southern Gate, Chichester,West Sussex, PO19 8SQ, UK: JohnWiley and Sons Ltd, 2017.

[29] lakdiva.org, "The 1945 proposal by arthur c. clarke for geostationary satellite communications." `http://lakdiva.org/clarke/1945ww/`, 2020. [Online; accessed apr 2020].

[30] A. C. Clark, "Extraterrestrial relays," *Wireless World*, 1945.

[31] J. A. Fraire, M. Feldmann, and S. C. Burleigh, "Benefits and challenges of cross-linked ring road satellite networks: A case study," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, 2017.

[32] J. A. Fraire, P. Madoery, S. Burleigh, M. Feldmann, A. C. J. Finochietto, N. Zergainoh, and R. Velazco, "Assessing contact graph routing performance and reliability in distributed satellite constellations," *HIndawi Journal of Computer Networks and Communications*, vol. 2017, no. 2830542, pp. 1–18, 2017. https://doi.or/10.1155/2017/2830542.

[33] J. R. Hall, M. Ghandehari, J. E. Schumann, and A. Golkar, "Starlink constellation design and operations," *SpaceOps Conference*, vol. April, 2020.

[34] "Oneweb." `https://www.oneweb.world/`. Accedido el 13 de abril de 2023.

[35] T. Arslan, N. Haridas, E. Yang, A. Erdogan, N. Barton, A. Walton, J. Thompson, A. Stoica, T. Vladimirova, K. McDonald-Maier, and G. Howells, "Espacenet: A framework of evolvable and reconfigurable sensor networks for aerospace-based monitoring and diagnostics," in *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, vol. 0, pp. 323 – 329, 07 2006.

[36] T. Vladimirova, X. Wu, and C. P. Bridges, "Development of a satellite sensor network for future space missions," in *2008 IEEE Aerospace Conference*, pp. 1–10, March 2008.

[37] R. Radhakrishnan, W. W. Edmonson, F. Afghah, R. M. Rodriguez-Osorio, F. Pinto, and S. C. Burleigh, "Survey of inter-satellite communication for small satellite systems: Physical layer to network layer view," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 4, pp. 2442–2473, 2016.

[38] J. Fraire, O. Jonckère, and S. Burleigh, "Routing in the space internet: A contact graph routing tutorial," *Journal of Network and Computer Applications*, vol. 174, p. 102884, 01 2021.

[39] C. Ji, X. Han, W. Dai, W. Ji, and Z. Wang, "Memory performance optimization of dtn relay node based on m/g/1," *Computer Communications*, vol. 177, pp. 24–32, 2021.

[40] C. Caini, H. Cruickshank, S. Farrell, and M. Marchese, "Delay- and disruption-tolerant networking (dtn): An alternative solution for future satellite networking applications," *Proceedings of the IEEE*, vol. 99, no. 11, pp. 1980–1997, 2011.

[41] S. E. Kady, M. H. Khater, and M. Alhafnawi, "Mips, arm and sparc-an architecture comparison," in *Proceedings of the World Congress on Engineering*, vol. 1, 2014.

[42] T. Lovelly and A. George, "Comparative analysis of present and future space-grade processors with device metrics," *Journal of Aerospace Information Systems*, vol. 14, pp. 1–14, 03 2017.

[43] R. Ginosar, "Survey of processors for space," *European Space Agency, (Special Publication) ESA SP*, vol. 701, 01 2012.

[44] B. Systems, "Rad750® radiation-hardened powerpc microprocessor." `https://www.baesystems.com/en-media/uploadFile/20210407062449/1434555668211.pdf`, 02 2019. [Online; accessed apr 2020].

[45] B. Systems, "Radiation-hardened electronics." `https://www.baesystems.com/en-us/product/radiation-hardened-electronics`, 2021. [Online; accessed apr 2020].

[46] "Freertos manual." `//https://www.freertos.org/FreeRTOS_Support_Forum_Archive/January_2012/freertos_xTaskGetTickCount_in_milliseconds_4956299.html`, 02 2012. [Online; accessed apr 2020].

[47] J. Kaur and S. R. N. Reddy, "Operating systems for low-end smart devices: a survey and a proposed solution framework," *International Journal of Information Technology*, vol. 10, pp. 49–58, Mar 2018.

[48] R. Barry, *Mastering the FreeRTOS™ Real Time Kernel A Hands-On Tutorial Guide*. Real Time Engineers Ltd. 20, 2016. Pre-release 161204 Edition.

[49] C. Holmberg, "P-access-network-info abnf update," rfc, RFC Editor, june 2016.

[50] S. Burleigh, "Contact graph routing." `https://datatracker.ietf.org/doc/html/draft-burleigh-dtnrg-cgr-00`, 2009. [Online; accessed apr 2020].

[51] J. Zhang, Q. HU, D. WANG, and W. XIE, "Robust attitude coordinated control for spacecraft formation with communication delays," *Chinese Journal of Aeronautics*, vol. 30, no. 3, pp. 1071 – 1085, 2017.

[52] S. M. Schilling K., "Ground station networks for distributed satellite systems," in *D'Errico M. (eds) Distributed Space Missions for Earth System Monitoring. Space Technology Library* (Springer, ed.), vol. 31, (New York, NY), pp. 355–371, 2013.

[53] M. Schmidt, R. S. Priya, and K. Schilling, "The pico-satellite uwe-1 and ip based telecommunication experiments," *IFAC Proceedings Volumes*, vol. 40, no. 7, pp. 721 – 725, 2007. 17th IFAC Symposium on Automatic Control in Aerospace.

[54] P. Muri and J. McNair, "Simulating delay tolerant networking for cubesats," in *Proceedings in The 5th Interplanetary CubeSat Workshop (iCubeSat), Boston, MA, USA*, 2012.

[55] S.-Y. Wang, "Distributed interplanetary delay/disruption tolerant network (dtn) monitor and control system," in *2012 IEEE Aerospace Conference*, pp. 1–9, 2012.

[56] J. Zhang, G. Wang, C. Liu, F. Zhao, and X. Zhang, "Delay tolerant network and the algorithms of DTN routing," *Journal of Physics: Conference Series*, vol. 1169, p. 012058, feb 2019.

[57] S. Grasic and A. Lindgren, "An analysis of evaluation practices for dtn routing protocols," in *Proceedings of the Seventh ACM International Workshop on Challenged Networks*, CHANTS '12, (New York, NY, USA), p. 57–64, Association for Computing Machinery, 2012.

[58] K. Scott and S. C. Burleigh, "Bundle Protocol Specification." RFC 5050, Nov. 2007.

[59] Consultative Committee for Space Data Systems (CCSDS), "Schedule-aware bundle routing (SABR) (blue book, recommended standard CCSDS 734.3-B-1." `https://public.ccsds.org/Pubs/734x3b1.pdf`, July 2019.

[60] J. P. A. León, "Evaluación del algoritmo contact graph routing en el caso de constelaciones satelitales leo," Master's thesis, Universidad de Buenos Aires, Buenos Aires, Sept. 2015.

[61] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture." `https://www.rfc-editor.org/rfc/rfc4838.txt`, 2007. [Online; accessed apr 2020].

[62] G. Araniti, N. Bezirgiannidis, E. Birrane, I. Bisio, S. Burleigh, C. Caini, M. Feldmann, M. Marchese, J. Segui, and K. Suzuki, "Contact graph routing in DTN space networks: overview, enhancements and performance," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 38–46, 2015.

[63] A. Vahdat and D. Becker, "Epidemic routing for partially-connected ad hoc networks," *Technical Report*, 06 2000.

[64] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, (New York, NY, USA), p. 145–158, Association for Computing Machinery, 2004.

[65] M. J. F. Alenazi, Y. Cheng, D. Zhang, and J. P. G. Sterbenz, "Epidemic routing protocol implementation in ns-3," in *Proceedings of the 2015 Workshop on Ns-3*, WNS3 '15, (New York, NY, USA), p. 83–90, Association for Computing Machinery, 2015.

[66] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," in *Service Assurance with Partial and Intermittent Resources* (P. Dini, P. Lorenz, and J. N. de Souza, eds.), (Berlin, Heidelberg), pp. 239–254, Springer Berlin Heidelberg, 2004.

[67] J. Xue, J. Li, Y. Cao, and J. Fang, "Advanced prophet routing in delay tolerant network," *Communication Software and Networks, International Conference on*, vol. 0, pp. 411–413, 01 2009.

[68] S. Han and Y. W. Chung, "An improved prophet routing protocol in delay tolerant network," *The Scientific World Journal*, vol. 2015, pp. 1–7, 01 2015.

[69] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: An efficient routing scheme for intermittently connected mobile networks," 2005.

[70] P. Das, P. Chowdhury, B. Poudel, and T. De, "Fibonary spray and wait routing in delay tolerant networks," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 6, pp. 3205–3216, 12 2016.

[71] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, "Maxprop: Routing for vehicle-based disruption-tolerant networks," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–11, 2006.

[72] E. Sammou and A. Abdali, "Routing in delay tolerant networks (dtn) —improved routing with maxprop and the model of "transfer by delegation" (custody transfer)," *International Journal of Communications, Network and System Sciences*, vol. 4, no. 1, pp. 53,58, 2011. doi: 10.4236/ijcns.2011.41006.

[73] A. Balasubramanian, B. Levine, and A. Venkataramani, "Dtn routing as a resource allocation problem," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, (New York, NY, USA), p. 373–384, Association for Computing Machinery, 2007.

[74] A. Balasubramanian, B. N. Levine, and A. Venkataramani, "Replication routing in dtns: A resource allocation approach," *IEEE/ACM Transactions on Networking*, vol. 18, no. 2, pp. 596–609, 2010.

[75] A. Balasubramanian, B. Levine, and A. Venkataramani, "Dtn routing as a resource allocation problem," *SIGCOMM Comput. Commun. Rev.*, vol. 37, p. 373–384, Aug. 2007.

[76] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: Social-based forwarding in delay-tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 11, pp. 1576–1589, 2011.

[77] S. C. Nelson, M. Bakht, and R. Kravets, "Encounter-based routing in dtns," in *IEEE INFOCOM 2009*, pp. 846–854, 2009.

[78] S. C. Burleigh, S. Farrell, and M. Ramadas, "Licklider Transmission Protocol - Specification." RFC 5326, Sept. 2008.

[79] C. Sacchi, K. Bhasin, N. Kadowaki, and F. Vong, "Technologies and applications of future satellite networking [guest editorial]," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 154–155, 2015.

[80] C. Dai, Q. Song, L. Guo, and Q. Chen, "Contact graph routing with network coding for leo satellite dtn communications," in *Space Information Networks* (Q. Yu, ed.), (Singapore), pp. 126–136, Springer Singapore, 2017.

[81] R. Diana, E. Lochin, L. Franck, C. Baudoin, E. Dubois, and P. Gelard, "A dtn routing scheme for quasi-deterministic networks with application to leo satellites topology," in *2012 IEEE Vehicular Technology Conference (VTC Fall)*, pp. 1–5, September 2012.

[82] S. Burleigh, "Contact Graph Routing," Internet-Draft draft-burleigh-dtnrg-cgr-01, Internet Engineering Task Force, July 2010. Work in Progress.

[83] J. Segui, E. Jennings, and S.Burleigh, "Enhancing contact graph routing for delay tolerant space networking," in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pp. 1–6, December 2011.

[84] E. Birrane, S. Burleigh, and N. Kasch, "Analysis of the contact graph routing algorithm: Bounding interplanetary paths," *Acta Astronautica*, vol. 75, pp. 108 – 119, 2012.

[85] C. Caini and R. Firrincieli, "Dtn for leo satellite communications," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, vol. 71, pp. 186–198, 01 2011.

[86] C. Caini and R. Firrincieli, "Application of contact graph routing to leo satellite dtn communications," in *2012 IEEE International Conference on Communications (ICC)*, pp. 3301–3305, 2012.

[87] N. Bezirgiannidis, C. Caini, D. D. P. Montenero, M. Ruggieri, and V. Tsaoussidis, "Contact graph routing enhancements for delay tolerant space communications," in *2014 7th Advanced Satellite Multimedia Systems Conf. and the 13th Signal Processing for Space Comms. Workshop (ASMS/SPSC)*, pp. 17–23, September 2014.

[88] N. Bezirgiannidis, C. Caini, and V. Tsaoussidis, "Analysis of contact graph routing enhancements for DTN space communications," *Int. Journal of Satellite Coms. and Networking*, vol. 34, no. 5, pp. 695–709, 2016.

[89] J. A. Fraire, P. Madoery, and J. M. Finochietto, "Leveraging routing performance and congestion avoidance in predictable delay tolerant networks," in *Wireless for Space and Extreme Environments (WiSEE), 2014 IEEE International Conference on*, pp. 1–7, October 2014.

[90] J. A. Fraire, P. Madoery, J. M. Finochietto, and E. J. Birrane, "Congestion modeling and management techniques for predictable disruption tolerant networks," in *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*, pp. 544–551, October 2015.

[91] S. Burleigh, C. Caini, J. J. Messina, and M. Rodolfi, "Toward a unified routing framework for delay-tolerant networking," in *2016 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pp. 82–86, 2016.

[92] G. Wang, S. C. Burleigh, R. Wang, L. Shi, and Y. Qian, "Scoping contact graph-routing scalability: Investigating the system's usability in space-vehicle communication networks," *IEEE Vehicular Technology Magazine*, vol. 11, pp. 46–52, December 2016.

[93] P. Madoery, P. Ferreyra, J. Fraire, F. Gomez, J. Barrientos, and R. Velazco, "Enhancing Contact Graph Routing Forwarding Performance for Segmented Satellites Architectures," in *1st IAA Latin American Symposium on Small Satellites*, (Argentina), March 2017.

[94] P. G. Madoery, J. A. Fraire, F. D. Raverta, J. M. Finochietto, and S. C. Burleigh, "Managing Routing Scalability in Space DTNs," in *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pp. 177–182, December 2018.

[95] N. Alesi, "Hierarchical Inter-Regional Routing Algorithm for Interplanetary Networks," Master's thesis, School of Engineering and Architecture, Department of Computer Science and Engineering, Bologna, Italy, 2018.

[96] P. Madoery, F. Raverta, J. Fraire, and J. Finochietto, "On the performance analysis of disruption tolerant satellite networks under uncertainties," in *Proceedings of the 2017 XVII RPIC Workshop*, September 2017.

[97] P. G. Madoery, F. D. Raverta, J. A. Fraire, and J. M. Finochietto, "Routing in space delay tolerant networks under uncertain contact plans," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2018.

[98] F. D. Raverta, R. Demasi, P. G. Madoery, J. A. Fraire, J. M. Finochietto, and P. R. D'Argenio, "A Markov Decision Process for Routing in Space DTNs with Uncertain Contact Plans," in *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pp. 189–194, December 2018.

[99] P. R. D'Argenio, J. A. Fraire, and A. Hartmanns, "Sampling distributed schedulers for resilient space communication," in *NASA Formal Methods - 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11-15, 2020, Proceedings (In Press)*, 2020.

[100] O. De Jonckère, "Efficient contact graph routing algorithms for unicast and multicast bundles," in *2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pp. 87–94, July 2019.

[101] S. Dhara, C. Goel, R. Datta, and S. Ghose, "CGR-SPI: A New Enhanced Contact Graph Routing for Multi-source Data Communication in Deep Space Network," in *2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pp. 33–40, July 2019.

[102] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture," RFC 4838, RFC Editor, April 2007.

[103] S. Burleigh, "Compressed Bundle Header Encoding (CBHE)," RFC 6260, RFC Editor, May 2011. `http://www.rfc-editor.org/rfc/rfc6260.txt`.

[104] R. D. (auth.), *Graph Theory*. Graduate Texts in Mathematics 173, Springer-Verlag Berlin Heidelberg, 5 ed., 2017.

[105] C. Secretariat and CCSDS, *Schedule-Aware Bundle Routing (SABR), White Book*. ccsds 232.0-b-2ed, 2016. ecommendation for Space Data System Standards.

[106] M. Demmer and K. Fall, "Dtlsr: Delay tolerant routing for developing regions," in *Proceedings of the 2007 Workshop on Networked Systems for Developing Regions*, NSDR '07, (New York, NY, USA), Association for Computing Machinery, 2007.

[107] J. A. Fraire, P. G. Madoery, and J. M. Finochietto, "On the design and analysis of fair contact plans in predictable delay-tolerant networks," *IEEE Sensors Journal*, vol. 14, pp. 3874–3882, Nov 2014.

[108] J. A. Fraire and J. Finochietto, "Routing-aware fair contact plan design for predictable delay tolerant networks," *Ad Hoc Networks*, vol. 25, pp. 303 – 313, 2015.

[109] J. A. Fraire, P. G. Madoery, and J. M. Finochietto, "Traffic-aware contact plan design for disruption-tolerant space sensor networks," *Ad Hoc Networks*, vol. 47, pp. 41 – 52, 2016.

[110] J. A. Fraire, P. G. Madoery, J. M. Finochietto, and G. Leguizamón, "Preliminary results of an evolutionary approach towards contact plan design for satellite dtns," in *Wireless for Space and Extreme Environments (WiSEE), 2015 IEEE International Conference on*, pp. 1–7, December 2015.

[111] P. G. Madoery, J. A. Fraire, and J. M. Finochietto, "Congestion management techniques for disruption-tolerant satellite networks," *International Journal of Satellite Communications and Networking*, vol. 36, no. 2, pp. 165–178, 2018.

[112] J. A. Fraire, G. Nies, C. Gerstacker, H. Hermanns, K. Bay, and M. Bisgaard, "Battery-aware contact plan design for leo satellite constellations:the ulloriaq case study," *IEEE Transactions on Green Communications and Networking*, pp. 1–1, 2019.

[113] D. Zhou, M. Sheng, X. Wang, C. Xu, R. Liu, and J. Li, "Mission aware contact plan design in resource-limited small satellite networks," *IEEE Transactions on Communications*, vol. 65, pp. 2451–2466, June 2017.

[114] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[115] GOMSPACE, *NanoMind A712C Datasheet*. GOMSPACE, 2 2013. Doc. ref: GS-DS-NM712C-1.1.

[116] ARM, *ARM7 TDMI (Rev3) Technical Reference Manual*. ARM.

[117] A. Kirillin, I. Belokonov, I. Timbai, A. Kramlikh, M. Melnik, E. Ustiugov, A. Egorov, and S. Shafran, "Ssau nanosatellite project for the navigation and control technologies demonstration," *Procedia Engineering*, vol. 104, pp. 97–106, 2015. Scientific and Technological Experiments on Automatic Space Vehicles and Small Satellites.

[118] C. Bridges, S. Kenyon, C. Underwood, and V. Lappas, "Strand-1: The world's first smartphone nanosatellite," in *2011 2nd International Conference on Space Technology*, pp. 1–3, 09 2011.

[119] L. K. Alminde, K. Kaas, M. Bisgaard, J. Christiansen, and D. Gerhardt, "Gomx-1 flight experience and air traffic monitoring results," in *28th Annual AIAA/USUConference on Small Satellite*, 2014.

[120] L. Kepko, C. Clagett, L. Santos, B. Azimi, D. L. Berry, T. M. Bonalsky, D. J. Chai, Matthew, Colvin, A. Cudmore, A. Evans, S. Hesh, S. L. Jones, J. A. Marshall, N. P. Paschalidis, Zach, Peterson, J. Rodriquez, M. Rodriquez, S. Sheikh, S. R. Starin, and E. Zesta, "Satellites ssc 17-iii-06 dellingr : Nasa goddard space flight center ' s first 6 u spacecraft," in *31st Annual AIAA/USUConference on Small Satellite*, 2017.

[121] "Memory management." `//https://www.freertos.org/a00111.html`, 02 2012. [Online; accessed apr 2021].