Facultad de Matemática, Astronomía,
Física y Computación

Universidad Nacional de Córdoba

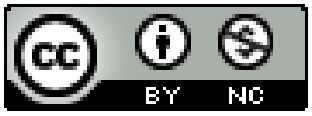
Estrategias de ruteos centralizados para redes espaciales tolerantes a demoras

Trabajo final para la Licenciatura en
Ciencias de la Computación

Elías Lihue Gasparini

Director:
Dr. Juan Andrés Fraire

Córdoba, Argentina 2020



Estrategias de ruteos centralizados para redes espaciales tolerantes a demoras

Autor: Elias Lihue Gasparini. Director: Dr. Juan Andrés Fraire

Diciembre 2020

Resumen

Las redes espaciales en órbita cercana a la tierra y en espacio profundo presentan características particulares respecto a las redes terrestres de Internet, lo que requiere de un abordaje distinto al problema del enrutamiento de datos. En particular, la dinámica orbital de los nodos, los recursos limitados en los satélites, las demoras e interrupciones en las transmisiones, la falta de conectividad parcial o total de origen a destino, entre otros, agregan una complejidad adicional a la hora de determinar y seleccionar una ruta. Por otro lado, la posibilidad de anticipar los cambios en la topología de la red, guiada por trayectorias predecibles de los nodos espaciales, habilita el uso de valiosa información para precomputar las rutas entre ellos.

En este marco, el protocolo Contact Graph Routing (CGR), desarrollado y mantenido por el laboratorio JPL, NASA, se enfoca en realizar el cálculo de las rutas a bordo de los satélites, y así permitir el uso de información actualizada para tomar decisiones de enrutamiento adecuadas. Sin embargo, las computadoras de a bordo cuentan con un reducido poder de cómputo y energía, lo que limita severamente la escalabilidad del enfoque actual en el estado del arte. Más aún, realizar un seguimiento riguroso y entender el por qué de las decisiones que toman los satélites en órbita de manera autónoma no es sencillo. Esto ha motivado a la industria espacial más conservadora a demandar soluciones alternativas.

En este trabajo se atiende esta demanda por medio de un análisis comparativo del enfoque distribuido de CGR y una primera versión centralizada del mismo. En base a esta, se estudia el cómputo en tierra de todas las rutas posibles que cada satélite pueda llegar a necesitar en un momento dado, relajando la necesidad de cómputo a bordo. La contraparte de esta propuesta es que el tiempo de cómputo puede resultar excesivo aún en tierra y que muchas de estas rutas posiblemente nunca se utilicen en órbita, lo cual resulta en una sobrecarga de la red, así como en un uso innecesario de memoria y energía. Mostraremos la necesidad de un procesamiento paralelizado para redes de tamaño considerable y discutiremos sobre el valor de una predicción acertada del tráfico futuro en la red. Dadas estas condiciones, el enfoque centralizado puede llegar a obtener ventaja en el aprovechamiento de los recursos sin la penalidad de una gran carga de almacenamiento. La contribución de este trabajo es, por lo tanto, una metodología acompañada de un conjunto de métricas que permiten medir este compromiso entre ambos enfoques. Además, se implementa una extensión de un simulador que permite analizar las contribuciones mencionadas a partir de su aplicación en diversos casos de estudios, algunos generados aleatoriamente y otros obtenidos de redes espaciales basadas en parámetros orbitales reales.

Summary

Spatial networks, such as Low-earth orbit and deep space networks, differ from those on earth in multiple aspects. Routing messages becomes a challenging problem in a dynamic ever-changing topology, where nodes are not expected to have stable connections, where both energy and storage capacity

are limited for most devices, and even no end-to-end connectivity is guaranteed. Routing algorithms can greatly benefit from the predictable orbit of nodes though.

The Contact Graph Routing (CGR) algorithm, elaborated by NASA, uses this valuable information to perform on-board computations and obtain efficient routes for packet delivery. However, the scalability of this approach is an open issue due to the satellites' limiting resources. Furthermore, keeping track of the routing decisions of the satellites can prove to be a difficult task when they perform it on-board and in an autonomous manner. These have motivated the spatial industry to seek alternative solutions.

In this project we aim to provide an alternative to the distributed version of CGR by simulating a centralized version, in which every route the nodes would ever need is computed in advance on earth, therefore reducing the necessity of running the algorithm on-board. The main drawback of this idea is that computing all routes in an arbitrary graph can be extremely time consuming, even with the processing power on earth. Moreover, providing an excessively large list of routes to a node can result in a waste of memory, as most of them might never be used. We will discuss the need for parallelized computing for medium-sized networks and an accurate prediction of the network traffic. Therefore, we present a set of metrics to evaluate the behaviour of both centralized and distributed approaches. In addition, we extend the implementation of a DTN simulator to support the centralized version of CGR and run simulations on different case studies, comprising random generated networks and realistic orbit parameter networks.

1. Introducción

El problema de enrutamiento de paquetes en redes terrestres, si bien es un problema de solución no trivial, es conocido y ha sido estudiado durante ya décadas. Es necesario tener en cuenta las propiedades de este tipo de redes para comprender qué dificultades se presentan a la hora de resolver el mismo problema en redes espaciales. Principalmente, el hecho de que el tiempo de respuesta de un mensaje (RTT, *Round Trip Time*) en la tierra esté en el orden de los milisegundos permite hacer suposiciones sobre el estado de conexión entre distintos nodos en la red y sobre la topología de la misma. En particular, para que dos entidades puedan conectarse e intercambiar datos es necesario que ambas tengan una conexión activa en todo momento durante la comunicación. Por ello, la demora en obtener una respuesta suele indicar congestión o falta de conexión de punta a punta. Además, esta misma característica de tener bajo RTT permite que: a) el proceso de enrutamiento en sí, es decir, encontrar la ruta óptima para un paquete entre un origen y un destino, se lleve a cabo al momento de envío del mensaje; b) las respuestas o mensajes de *feedback* sean rápidas; y c) los nodos intermedios en una conexión tengan solamente la responsabilidad de enrutar los paquetes recibidos de forma óptima. Esto es, un paquete puede descartarse una vez reenviado. Es deber del nodo de origen mantener un registro de los paquetes emitidos y correctamente recibidos y, en el caso de extravío de un mensaje, generar uno nuevo. Por otro lado, la topología en las redes terrestres es estable. Los cambios a gran escala (excluyendo las conexiones y desconexiones de nodos finales) no son frecuentes y los canales de transmisión son bidireccionales con una tasa de datos simétrica. Por último, y no de menor importancia, la energía para todos los nodos en la red es, a fines prácticos, ilimitada.

Ahora bien, en las redes espaciales, conformadas tanto por nodos espaciales como nodos terrestres (antenas transmisoras y receptoras de mensajes), es natural el constante cambio de topología. El movimiento de los satélites y los movimientos de rotación y traslación de la Tierra llevan a que los nodos en la red entren y salgan permanentemente del rango de transmisión. A esto se le suman las posibles oclusiones, desde un tren que pasa por un túnel hasta un planeta que se interpone entre dos nodos. A su vez, el tiempo de propagación de la señal puede ser de segundos, minutos u horas para llegar a destino. Sobre esta base es que se propone el esquema de DTN (*Delay/Disruption Tolerant Networking*), redes tolerantes a demoras y a disrupciones, en donde no se asume una respuesta instantánea por parte del receptor. Sobre esta arquitectura, se plantea el Protocolo Bundle (BP, *Bundle Protocol*) [1], el cual brinda

servicios a la capa de aplicación y se asienta sobre otros protocolos de internet que garanticen la conectividad entre dos nodos (no necesariamente todo par de nodos deben estar conectados por el mismo protocolo de internet) (ver Fig. 1). El paradigma del BP es *store, carry and forward*, en donde cada nodo enrutador de la red almacena los paquetes en tránsito hasta tener la posibilidad de enviarlos hacia el siguiente destino, brindando la capacidad de comunicación sin tener una garantía de conexión punta a punta. Entre los problemas que surgen en las DTN por no poder utilizar algoritmos clásicos, basados en un bajo RTT y conexiones estables, como ser privacidad, seguridad, control de congestión, QoS, nos enfocaremos en el enrutamiento de paquetes.

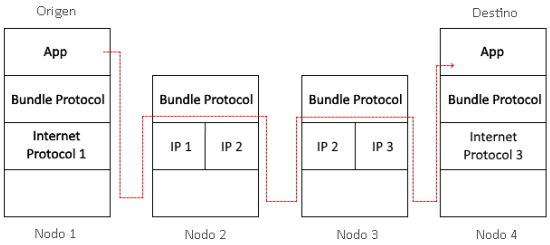


Figura 1: Nodos conectados por el Protocolo Bundle, no necesariamente bajo el mismo protocolo de internet.

El resto del documento se oraniza como se sigue: en la sección 2 se introducen los conceptos de DTN-predecibles y contactos, mientras que se explican los aspectos relevantes a enrutamiento de CGR en la sección 3. Dos posibles algoritmos de búsqueda de rutas para la versión centralizada de CGR son detallados en la sección 4. Las secciones 5 y 6 muestran los casos de estudios y los resultados para los mismos, respectivamente. Por último, abarcamos en la sección 7 nuestra conclusión y posibles trabajos futuros.

2. Delay-Tolerant Network

En general, las redes que presentan la característica de constantes desconexiones entre sus nodos caen en la categoría de DTN. Este esquema es aplicable a numerosos escenarios [2], no solo en redes espaciales, sino también en redes militares, redes de sensores esparsos (como ser redes subacuáticas), situaciones de catástrofes naturales o terrenos de dificultosa conectividad. En muchos de ellos, las oportunidades de transmisión, ya sean frecuentes o no, ocurren de manera aleatoria (o, al menos, de manera no predecible) y las conexiones de punta a punta son poco probables. Los algoritmos clásicos de enrutamiento fallan en encontrar una ruta apropiada en estos casos. Por ello, surgen diferentes aproximaciones para lograr la entrega de mensajes en DTNs. Por un lado, algoritmos de inundación han sido probados [3], con una esperable robustez y baja eficiencia en la utilización de recursos de los nodos. Por otro, el algoritmo de *Spray and wait* [4] inserta en la red a lo sumo L copias del mensaje con una heurística para repartirlos, obteniendo una probabilidad alta de entrega con un consumo menor que el de inundación, mientras que el algoritmo PRoPHET [5] se enfoca en las DTN en donde los nodos no circulan de manera completamente aleatoria e intenta aprovechar la probabilidad de futuros encuentros dada la información de contactos previos.

Una ventaja con la que se corre en las redes espaciales es que las trayectorias tanto de la Tierra como la de los satélites son conocidas y los cambios en la topología de la red pueden ser anticipados con semanas de antelación. Este tipo de DTNs, en donde las oportunidades de transmisión son previsibles, se denominan "DTN predecibles". En adelante nos enfocaremos solo en estas. Vale aclarar que cuando la predicción de las conexiones entre nodos y la calidad de transmisión es certera, la utilización de esta información para enrutamiento resulta en un menor tiempo de entrega de los mensajes y mejor aprovechamiento de los recursos [6].

Los contactos, concepto fundamental en las DTN, son esencialmente oportunidades de transmisión de información entre dos nodos en la red (fig. 2). Un contacto $C_{A,B,d}^{t1,t2,R}$ está definido por un tiempo de inicio t1 y tiempo de fin t2 en donde el nodo A podrá transmitir datos al nodo B con una tasa de transferencia R y una distancia de d segundos luz entre ellos.

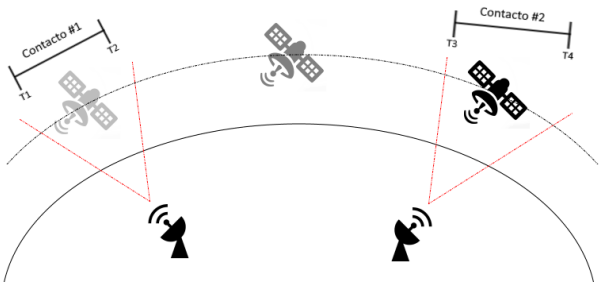


Figura 2: Ejemplos de contactos

Denotaremos a t1 como $start(C)$, a t2 como $end(C)$, al nodo A como $source(C)$, a B como $dest(C)$ y a la demora de propagar la señal por d como $delay(C)$. Notar que múltiples contactos pueden existir entre el mismo par de nodos, ya sea por distintos intervalos de tiempo disjuntos, o bien intervalos no disjuntos, pero con una tasa distinta de transferencia (como ser el caso de dos medios físicos de transferencia distintos). Una *ruta* será entonces una sucesión finita de contactos C_i, $i > 0$ en donde se cumple

$$\forall i > 0. dest(C_{i-1}) = source(C_i)$$

$$\wedge$$

$$\forall i, j. end(C_i) < start(C_j) \implies i < j$$

Para cada ruta se puede calcular su *tiempo de entrega*, es decir, el tiempo que demoraría el primer *byte* de datos en llegar de origen a destino. Esto es: dada una ruta S conformada por N contactos $C_1, ..., C_N$, $tiempo\_de\_entrega(S) = T_S(N)$, donde T_S se define recursivamente como se sigue:

$$T_S(1) = start(C_1) + delay(C_1)$$

$$T_S(i) = max(T_S(i-1), start(C_i)) + delay(C_i),$$
$$\forall i. 1 < i <= N$$

Intuitivamente, si queremos transmitir un paquete desde el nodo i hasta el nodo $i+1$ a través de un contacto C, la demora de propagación va a estar limitada por la demora hasta el nodo i o bien por el comienzo del contacto C. A esto se le suma la demora de propagación $delay(C)$. De manera similar, se define el *tiempo límite* de una ruta como el menor tiempo de finalización de sus contactos, teniendo en cuenta la demora de propagación hasta estos. Sea nuevamente S compuesta por $C_1, ..., C_N$, $tiempo\_limite(S) = L_S(N)$, donde L_S queda definida por

$$L_S(1) = end(C_1)$$

$$L_S(i) = min(L_S(i-1), end(C_i) - D_S(i-1)$$
$$\forall i. 1 < i <= N$$

y $D_S(i)$ indica la demora de propagación acumulada hasta el i-ésimo contacto.

$$D_S(1) = delay(C_1)$$

$$D_S(i) = D_S(i-1) + delay(C_i),\ \forall i. 1 < i <= N$$

Cualquier paquete asignado a esta ruta deberá comenzar a transmitirse antes de su tiempo límite para tener posibilidad de llegar a destino. Luego de este plazo, la ruta será declarada inválida.

En los escenarios que se plantean en la sección 5, las conexiones ocurren por "capas.º "jerarquías", en las que nodos pertenecientes a un nivel solo pueden conectarse con nodos en un nivel inmediatamente superior o inferior. Por ejemplo: una red en la que antenas de espacio profundo (DSN antennas) no puedan establecer una comunicación directa con robots o sensores en Marte y deban hacerlo por medio de satélites que orbitan en este planeta; o bien una red terrestre en la que, para comunicarse con nodos aislados por falta de infraestructura, se necesiten satélites de órbita cercana a la Tierra (LEO, *Low earth orbit*) o algún transmisor intermediario. Los nodos en un mismo nivel pueden o no tener conexión entre ellos. Cuando los satélites tienen la posibilidad de enviarse paquetes de manera directa, se dice que existen conexiones intersatelitales (ISL, *Inter-satellite Links*).

3. Contact Graph Routing

Todas las conexiones y cambios previstos en la topología pueden verse como un conjunto de contactos, los cuales pueden ser plasmados en lo que se denomina un *plan de contactos*. Algoritmos que no utilizan esta información a su favor e intentan descubrir rutas en tiempo real y de manera oportunística muestran, como es esperable, un bajo rendimiento [7]. El algoritmo de *Contact Graph Routing* (CGR) [8], por su parte, propone que los nodos exploren de forma autónoma el plan de contactos y encuentren rutas óptimas para el enrutamiento de paquetes. Aquí "ruta óptimaïmplica una preferencia sobre una métrica o un conjunto de métricas. En este caso, se busca la ruta con menor tiempo de entrega, otras alternativas siendo menor cantidad de nodos en la ruta (menor cantidad de "saltos"), menor probabilidad de pérdida del mensaje, mayor volumen de transmisión, entre otras. Con respecto al cómputo del camino de menor costo entre dos nodos, las primeras versiones de CGR omiten el uso del grafo que surge naturalmente al representar una DTN: un multigrafo dirigido cuyas aristas varían en función del tiempo. En cambio, se propone correr el algoritmo de camino de costo minimal de Dijkstra sobre el *grafo de contactos*, y de aquí el nombre de CGR, que se construye de la siguiente manera: a) Un nodo por cada contacto en el plan de contactos; b) una arista del contacto C_i al contacto C_j si se cumple

$$dest(C_i) = source(C_j)$$

$$\wedge$$

$$start(C_i) + delay(C_i) < end(C_j)$$

Hasta aquí, ha de notarse que la definición de las aristas en el grafo de contactos no depende del tiempo. Tanto $start(C_i)$, $end(C_j)$ y $delay(C_i)$ son fijos y conocidos por el plan de contactos. Además, encontrar un camino minimal en este grafo es equivalente a encontrar una sucesión de contactos C_i, $i > 0$ que cumple la condición de ser una ruta válida. Queda por determinar el costo de cada arista en el grafo de contactos. Puesto que la definición de T dada anteriormente es recursiva, es necesario contar con un caso base, un contacto inicial a partir del cual calcular los costos sucesivos. Para ello, se requiere fijar el nodo de origen de la/s ruta/s por computar. En otras palabras, el costo de las aristas en el grafo de contactos depende del nodo de partida. Sea A el nodo de origen, se define un "contacto virtual"$C_0 = C^{0,\infty,\infty}_{A,A,0}$, introducido únicamente durante el proceso para encontrar rutas. Se obtiene así un nuevo nodo en el grafo de contactos con aristas de costo nulo hacia todos los contactos C_i tales que $source(C_i) = A$. Luego, para una ruta cualquiera S_0 formada por los contactos $C_0, C_1, ..., C_N$, fácilmente se construye la ruta equivalente S_1 con $C_1, ..., C_N$, ya que $source(C_1) = A$ y $tiempo\_de\_entrega(S_0) = tiempo\_de\_entrega(S_1)$.

El diseño del plan de contactos es un área en sí misma y no nos enfocaremos en ello en este trabajo. Sin embargo, es importante saber que se deben tener ciertas consideraciones durante su armado. La predictibilidad de la ubicación de los nodos no es suficiente para determinar si un contacto será efectivo. Los atributos de comunicación de los satélites, como ser la potencia de transmisión, la modulación y codificación de la transmisión, la probabilidad de error, y la orientación de su/s antena/s en un momento particular, son factores claves para decidir por la factibilidad de un contacto. Más aún, las limitaciones de *hardware* pueden llevar a que, de dos o más posibilidades de conexión en un período de tiempo, solo una sea utilizable. El lector puede referirse a [9] para más detalles sobre el diseño e implementación de los planes de contactos.

Por otro lado, el Protocolo Bundle cuenta con ciertos aspectos relevantes al enrutamiento. Entre ellos se encuentran los *Bloques de extensión* [1]. La especificación del protocolo permite definir, para una implementación particular del BP, bloques de información adicional que se agregarán al encabezado de los mensajes. El documento prevé la posibilidad de que distintos nodos bajo la ejecución del BP, en tanto cumplen con sus especificaciones, entiendan parcialmente los bloques de extensión presentes en un encabezado. En particular, el bloque de extensión *Source Routing* [10] permite al nodo de origen insertar la ruta encontrada para entregar el paquete. De esta manera, los nodos subsiguientes requerirán únicamente hacer una comprobación de que la ruta descripta en

el bloque de extensión siga siendo válida, en vez de correr el algoritmo de enrutamiento. Esto disminuye considerablemente el tiempo de procesamiento total utilizado para esta tarea.

3.1. Heurísticas en CGR

Si bien ya se tiene definido el grafo de contactos, es necesario conocer cuándo y cómo los nodos siguiendo el BP ejecutan el algoritmo de enrutamiento. Es importante recordar que dos ejecuciones del algoritmo de Dijkstra sobre el mismo grafo resolverán en rutas idénticas. Esto nos dice que el grafo de contactos deberá ser modificado entre ejecución y ejecución. A medida que el tiempo transcurre, ciertos contactos en el plan de contacto se vuelven obsoletos debido a su tiempo de finalización. Por lo tanto, al buscar rutas, se deben omitir todos los contactos con tiempo de finalización menor al tiempo actual (momento de ejecución del algoritmo). Por otro lado, no toda ruta con mejor tiempo de entrega será útil para transmitir un mensaje. Cada paquete enrutado es asignado a una ruta, en la cual se "reserva" un cierto volumen de transmisión de acuerdo con el tamaño del paquete. Dado que todas las rutas están limitadas en su volumen de transmisión, es posible que la mejor de ellas no tenga capacidad *remanente* suficiente (volumen no reservado) para soportar el envío. En este caso, deberá modificarse el plan de contactos de acuerdo a algún criterio de selección de contacto para encontrar una nueva ruta. Con respecto a "cuándo ejecutar la búsqueda de rutas", existen dos enfoques: 1) Únicamente computar rutas de acuerdo con la demanda de tráfico en la red. Es decir, ante la llegada de un paquete en que no se cuenta con rutas válidas para la transmisión, buscar rutas hasta hallar una o más que soporten el envío. 2) Anticiparse a la demanda de tráfico hacia un nodo en particular y pre-computar rutas de tal manera que estén disponibles en el momento de arribo de un mensaje. Está claro que el segundo enfoque requiere de una mayor utilización de recursos, tanto en procesamiento como en almacenamiento para las rutas. Más aún, las rutas precomputadas pueden resultar ser no utilizadas en absoluto, lo cual significa un gasto innecesario de energía. Sin embargo, una anticipación acertada es de mucha utilidad, ya que disminuye el tiempo que un paquete reside en un nodo hasta ser enrutado y aumenta su posibilidad de tomar una ruta limitada por una finalización próxima en el tiempo.

Algunos de los criterios propuestos para la forma de ejecución distribuida de CGR para modificar el plan de contactos a medida que se buscan rutas se listan a continuación:

Anchor contact A medida que se encuentran rutas, marcar el primer contacto en la ruta como *anchor contact* (contacto de anclaje) y eliminar temporalmente el contacto limitante en el tiempo de finalización. Para las búsquedas subsiguientes, si el *anchor contact* difiere del nuevo contacto inicial, cambiar de *anchor contact*, eliminar permanentemente el *anchor contact* anterior, y restaurar los contactos eliminados temporalmente. De esta manera, se agotan todos los contactos que tienen como partida el nodo de origen. La heurística de *anchor contact* fue utilizada en las versiones iniciales de CGR y posteriormente abandonada, puesto que falla en encontrar casos en los que las mejores rutas alternan de contacto inicial.

First ended De la última ruta encontrada, tomar el contacto con menor tiempo de finalización (determinante del tiempo límite de la ruta) y eliminarlo del grafo de contactos. Notar que esto no es suficiente para garantizar que las rutas encontradas a continuación tengan un tiempo de finalización mayor.

First depleted De la última ruta encontrada, eliminar del grafo de contactos el contacto con menor capacidad de transmisión. Esta estrategia es útil si se la utiliza buscando rutas por demanda de tráfico. Esto es porque una ruta deja de ser útil por tiempo (caso en el que los contactos finalizados serán filtrados naturalmente al buscar una nueva ruta), o bien por un volumen de transmisión agotado o reservado. De darse este último caso, se filtra el contacto con menor capacidad de transmisión y se corre el algoritmo de Dijkstra nuevamente.

Algoritmo de Yen Siendo el estado del arte en CGR, el algoritmo de Yen [11] logra encontrar las

mejores K rutas acíclicas de un nodo A a un nodo B en un grafo ponderado, dirigido y sin ciclos negativos. Es importante mencionar que, a diferencia de las heurísticas anteriores, toma una aproximación no *greedy* para filtrar contactos del plan de contactos. Aún así, su complejidad aumenta linealmente con el valor de K.

4. CGR centralizado

El principal problema de correr CGR de manera distribuida, en donde cada nodo tiene que procesar el plan de contactos para poder enrutar un paquete, es el limitante poder de cómputo con que cuentan los satélites. Los procesadores espaciales son construidos con consideraciones adicionales a los terrestres, que respectan a los niveles de radiación a los que estará expuesto el dispositivo y a su consumo energético. Esto lleva a velocidades de procesamiento considerablemente menores[1]. A su vez, la tarea de enrutar paquetes se vuelve más costosa mientras más alto es el tráfico en la red, ya que más veces deberá ejecutarse el algoritmo, y más cuando se tiene un plan de contactos compuesto por decenas de miles de contactos.

En este trabajo se propone liberar a los nodos espaciales de tal responsabilidad, proveyéndolos con una *lista de rutas precomputadas en tierra*. Dicha lista estará conformada por todas las rutas que el nodo pueda necesitar en un futuro cercano. A medida que estas quedan obsoletas, será responsabilidad de algún nodo central en tierra reprocesar y transmitir una nueva lista actualizada. Ahora bien, enumerar todos los caminos de un nodo a otro en un grafo arbitrario es un problema cuyo tamaño de respuesta no es polinomial con respecto al de la entrada (*input*). Dada la magnitud del grafo de contactos, sin una heurística para recortar la cantidad de rutas encontradas, el problema puede volverse inabarcable, incluso con el poder de cómputo terrestre. Para ello, proponemos y comparamos el desempeño de dos algoritmos centralizados encargados de producir dicha lista de rutas.

[1]La industria espacial es pequeña y costosa en comparación con las de tecnologías terrestres. Los avances tecnológicos para componentes espaciales son más lentos. Muchos teléfonos móbiles ya han alcanzado velocidades de procesamiento muy superiores a satélites actuales.

4.1. *First Ended* Centralizado

Algoritmo 1: First-ended centralizado

Entrada: Plan de contactos CP, Conjunto de nodos en la red $\mathcal{S}$
Resultado: Lista de rutas por nodo de origen

```
1  Resultado ← {}
2  foreach Nodo de origen O en S do
3  |  Resultado[O] ← {}
4  |  Nuevas_rutas ← Dijkstra(CP, O, S)
5  |  while Nuevas_rutas ≠ ∅ do
6  |  |  Resultado[O] ← Resultado[O] +
   |  |   Nuevas_rutas
7  |  |  C ← menor_contacto(Nuevas_rutas)
8  |  |  eliminar_contacto(C, CP)
9  |  |  Nuevas_rutas ← Dijkstra(CP, O, S)
10 |  CP ← restablecer_plan_de_contactos()
```

Similar a la heurísica que se plantea en la sección 3.1, encontrar rutas óptimas será por medio del algoritmo de Dijkstra, eliminando entre ejecución y ejecución el contacto utilizado con menor tiempo de finalización. La principal diferencia será no trabajar sobre el grafo de contactos para ejecutar el algoritmo, sino el multigrafo en donde cada nodo se corresponde con un nodo en la red y en donde las aristas son justamente contactos, conexiones entre dos nodos. Las adaptaciones del algoritmo de Dijkstra para funcionar en un multigrafo son sencillas [12]. Basta considerar todas las aristas entre cada par de nodos y, para reconstruir la ruta, registrar no el nodo predecesor, sino la arista utilizada para lograr el camino. Por otro lado, se aprovechará el hecho de que una sola ejecución del algoritmo de Dijkstra permite obtener la mejor ruta hacia todo otro nodo en el grafo (árbol de costo minimal). Referirse a Fig 3.
Si bien en la versión distribuida se permite terminar el algoritmo tempranamente una vez alcanzado el nodo destino del paquete, en el CGR centralizado dejaremos que el algoritmo concluya naturalmente una vez visitados todos los nodos, con el fin de disminuir la

cantidad de ejecuciones totales. Un pseudo-código del algoritmo puede encontrarse en Alg. 1. El algoritmo de Dijkstra en multigrafo se encuentra en el apéndice al final del documento, sección 8.1. La métrica de las rutas por optimizar seguirá siendo el tiempo de entrega. El costo de las aristas, como se mencionó en la sección 3, será determinado en el momento de análisis y selección de aristas, ya que depende del tiempo de entrega del nodo siendo visitado. Notar que esto no aumenta la complejidad del algoritmo de Dijkstra, puesto que asignar el costo a una arista tiene complejidad $\mathcal{O}(1)$. Durante la fase de eliminación de contactos, se tomará como referencia al contacto C de menor tiempo de finalización entre todos los contactos constituyentes de las rutas encontradas y se lo eliminará del conjunto de aristas. Cabe aclarar que el algoritmo de Dijkstra puede construir rutas halladas en ejecuciones previas, ya que la eliminación de un contacto no garantiza la necesidad de nuevas rutas para todo par de nodos. Esto dice que el conjunto de rutas finales debe ser filtrado por rutas duplicadas.

4.2. BFS

Algoritmo 2: BFS centralizado

Entrada: K, M, tiempos de filtro T, plan de contactos CP, conjunto de nodos en la red $\mathcal{S}$

Resultado: Lista de rutas por nodo de destino

```
1 Resultado ← {}
2 foreach Nodo de origen O ∈ S do
3 |   Resultado[O] ← {}
4 |   foreach t ∈ T do
5 |   |   filtrar_contactos_por_tiempo(CP, t)
6 |   |   Rutas ← BFS(K, M, CP, O, S)
7 |   |   Resultado[O] ← Resultado[O] + Rutas
```

Como alternativa al algoritmo centralizado anterior, proponemos una forma de explorar el grafo de contactos para obtener las mejores K rutas para cada tiempo $t \in T$, donde T es un conjunto arbitrario de tiempos. Luego de cada iteración, todo contacto con tiempo de finalización menor a t será filtrado

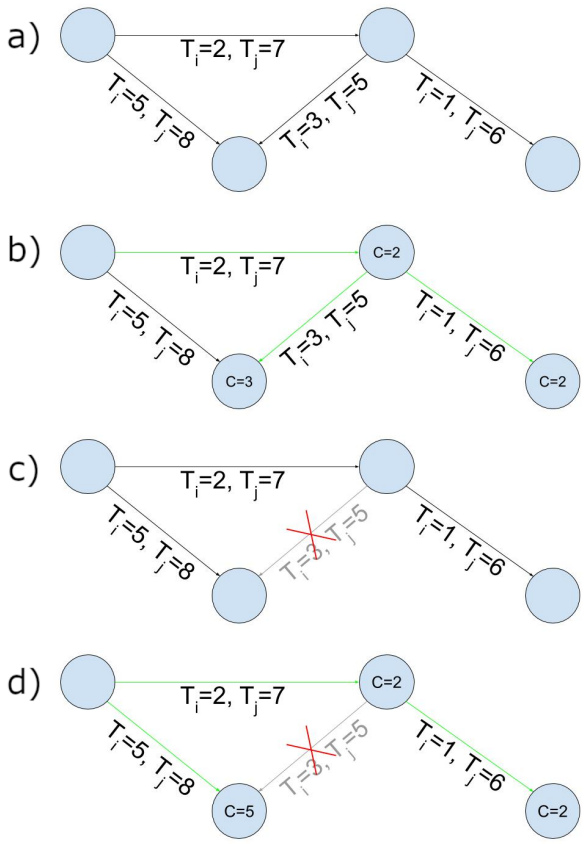


Figura 3: Primeras dos iteraciones del algoritmo *First ended*. a) pequeño grafo de ejemplo. b) primer árbol de costo minimal obtenido con el algoritmo de Dijkstra. Se encuentran 3 rutas, una por cada destino. c) eliminación del contacto utilizado con menor tiempo de finalización. d) nuevo árbol minimal. Se encuentra una ruta nueva.

del plan de contactos. Utilizaremos una aproximación de fuerza bruta, limitando el espacio de búsqueda para que la cantidad de rutas encontradas y el tiempo de búsqueda sean de uso práctico. La búsqueda consistirá en una exploración a modo BFS (*Breath-first search*), en la que se permita más de una visita por nodo. El pseudocódigo del algoritmo puede verse en Alg 2. Las rutas se irán construyendo con cada paso que expande la búsqueda. Diremos que una ruta R puede ser extendida con un contacto C si $tiempo\_de\_entrega(R) < end(C)$. De esta manera, se obtiene una nueva ruta R' hacia el destino $dest(C)$. Por otra parte, será necesario evitar ciclos en las rutas, ya que la exploración permite volver a visitar nodos. Esto se logra fácilmente debido a que se pueden obtener los nodos participantes en cada ruta a costo de una búsqueda lineal sobre los saltos de la ruta, incrementando la complejidad de timepo. En la figura 4 se muestra una iteración del BFS con K=2. En cambio, si el parámetro hubiese sido K=1, la ruta R1 debería ser eliminada al encontrar R3, ya que esta última tiene mejor tiempo de arribo. Adicionalmente, incluimos un parámetro M para limitar la cantidad de saltos (longitud de las rutas).

Si bien los resultados en la sección 6 se toman sin límite en el parámetro M, en la práctica puede ser útil para lograr una reducción del consumo de energía, evitando emisiones y recepciones innecesarias.

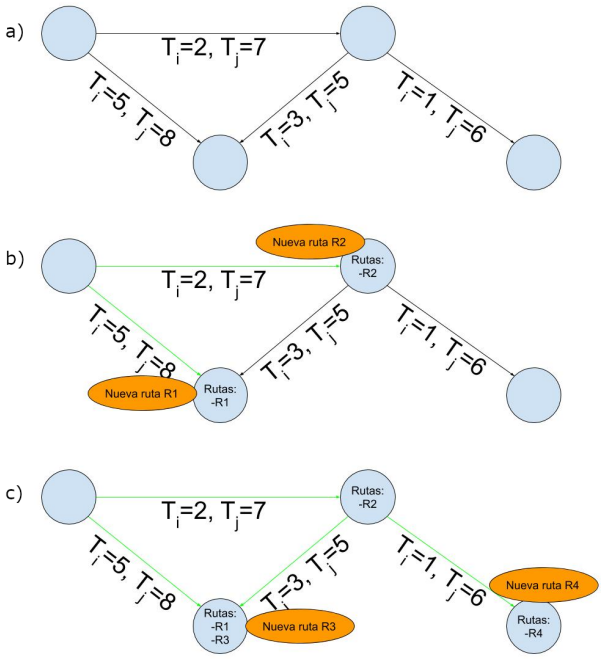


Figura 4: Primera iteración del algoritmo BFS, K=2. a) grafo de ejemplo. b) primera expansión desde el nodo inicial. Se agregan las rutas encontradas. c) Expansión desde un nodo vecino. Al permitir dos rutas por cada posible destino (parametro K), se almacenan las dos encontradas. De haber sido K=1, se almacenaría solo la mejor (R3).

5. Casos de estudio

5.1. Tráfico de la red

Debido a que la demora total del algoritmo distribuido es dependiente de la cantidad de paquetes enrutados (aproximadamente la cantidad de veces que se ejecutará la búsqueda de rutas), se determinaron tres frecuencias de envío de paquetes de datos (*bundles*), a saber: tráfico alto, tráfico medio y tráfico bajo. Estos intervalos se corresponden con los siguientes valores:

- Tráfico alto: 1 *bundle* por hora
- Tráfico medio: 1 *bundle* cada 2 horas
- Tráfico bajo: 1 *bundle* cada 3 horas

Para las versiones centralizadas, se utilizó la misma frecuencia que la de tráfico alto con el fin de comparar métricas sobre el envío de paquetes. En todos los casos, los paquetes fueron generados para telemetría, es decir, para reportar el estado de salud de cada nodo en la red. Estos paquetes son generados en intervalos regulares de tiempo y no tienen mayor significancia en tamaño. Asumimos para este modelo una congestión nula en la red y dejamos el análisis con otras figuras de tráfico para trabajos futuros. Además, para asegurarse de que la cantidad de veces que se enruta un paquete no dependa de la longitud de la ruta (la cantidad de veces que un paquete es recibido y reenrutado por nodos intermedios), se utilizó el bloque de extensión de *Source Routing*, como se mencionó en la sección 3.

5.2. Escenarios

Cada red es descripta por un único plan de contactos. No se consideraron casos en los que este quede obsoleto y haya que actualizarlo. El fin de la simulación es determinado por el período abarcado por el plan de contactos. Notar que cuanto más largo sea este período, más paquetes de telemetría serán enviados. Los distintos planes de contacto presentan diferencias en la cantidad de nodos involucrados en la red, así como la cantidad de contactos entre ellos. En todos los casos se consideró que las conexiones entre nodos se generen de manera jerárquica y sin ISL. Adicionalmente, se agrega en cada plan de contactos un nodo central en tierra, llamado Centro de Control de Misión (CCM), con conexiones entrantes de todos los nodos del primer nivel, al cual estarán direccionados los paquetes de telemetría. A continuación se describe cada uno de los escenarios utilizados:

Walker Esta red, como un ejemplo más de las *Ring Road Networks* (RRNs) [13], consiste en 122 nodos organizados de la siguiente manera: a) 10 estaciones terrestres (GS), b) 12 satélites LEO y c) 100 nodos terrestres (GN). Es en este mismo orden es en que se obtienen las jerarquías (GS-sat-GN). El plan de contactos abarca 24hs, consta de aproximadamente 9.700 contactos, y fue generado basado en parámetros orbitales reales obtenidos de la herramienta STK[2]. Una ilustración puede verse en la figura 5.

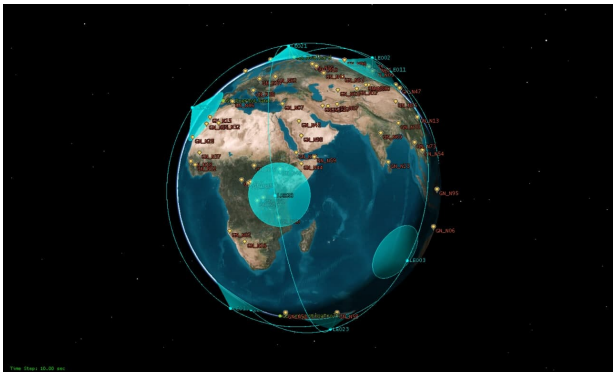


Figura 5: Visualización del escenario *Walker*.

Lunar Ring Road Siendo un posible diseño de una red con el objetivo de extraer datos de la cara oculta de la Luna [14], la misma se compone de 23 nodos: a) 3 antenas DSN, b) 4 estaciones lunares, c) 8 satélites orbitando la Luna, y d) 8 sensores lunares (visualización en figura 6). La duración del plan de contactos se extiende por 2 semanas y contiene más de 10.000 contactos. Este escenario también es basado en dinámicas orbitales realistas generadas con STK[3].

Escenarios aleatorios Para obtener más información sobre el comportamiento de estos algoritmos, se implementó un generador de casos aleatorios que permita variar parametros de la red. Se dividieron los escenarios en dos conjuntos: (a) Set 1: variando entre 3 valores posibles para la cantidad de nodos y 5 valores para la cantidad de contactos, se establecieron 15 combinaciones de parámetros. Con la idea de promediar métricas, se generaron 10 escenarios por cada combinación, para un total de 150 escenarios. Los parámetros son como se sigue. Número de nodos: 33, 48, 63. Contactos: 5000, 7000, 9000, 11000, 13000.

[2]Los planes de contactos, archivos de escenario de STK y visualizaciones animadas pueden obtenerse en `https://sites.google.com/unc.edu.ar/dtsn-scenarios`

[3]Mirar nota al pie 2.

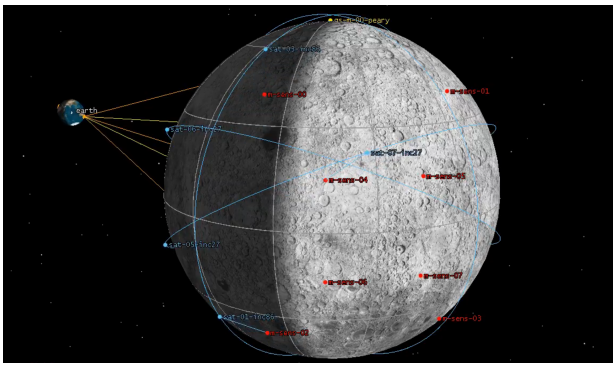


Figura 6: Visualización del escenario lunar.

Jerarquías: 3-X-X-X, donde X=10,15,20 (respectivamente a la cantidad de nodos). Duración del plan de contactos: 1 día. (b) Set 2: 7 valores posibles para el período abarcado por el plan de contactos, de 1 a 7 días. Para este conjunto se fijó el número de nodos en 33 y la cantidad de contactos en un promedio de 3000 por día. En ambos conjuntos (a) y (b), las redes fueron establecidas como *full-duplex*, en donde las conexiones son bidireccionales (dos contactos por cada conexión).

6. Análisis de resultados

Para obtener métricas y realizar comparaciones entre la versión distribuida y las dos versiones centralizadas, se realizaron simulaciones con la herramienta DTNsim [15], desarrollada sobre el framework Omnet++. En todos los casos de estudio, se utilizó la heurística *first depleted* para la forma de enrutamiento distribuida, ya implementada en el simulador, con una búsqueda de rutas únicamente bajo demanda de tráfico. Se le dió soporte al algoritmo centralizado implementándolo en un nuevo módulo[4] para el DTNsim. Además, se optimizó el código relacionado a la búsqueda de rutas y operaciones en el plan de contactos, como ser la enumeración de vecinos, para disminuir la complejidad total de los algoritmos. Dejamos al pie de página el enlace hacia la "*notebook*" utilizada[5] para procesar los resultados y graficar.

Dado que en la versión distribuida cada nodo corre la búsqueda de rutas de manera independiente del resto, todas las métricas producidas por nodos espaciales (nodos bajo el análisis de consumo) fueron agregadas con una sumatoria, representando un costo general. Se incluye una segunda escala por nodo en el margen derecho de cada gráfico. Adicionalmente, a partir de las velocidades de procesamiento en [16], se sacó un coeficiente para estimar la diferencia entre un procesador terrestre y uno espacial e incluirlo al comparar el tiempo de cómputo de las rutas. Las simulaciones se corrieron sobre un procesador Intel Core i7-4790, llegando a 6.80 GFLOPS (*single core*). Siendo que el procesador espacial Proton 200k SBC alcanza alrededor de 900 MFLOPS, se obtiene así un factor de

$$\frac{6{,}80 \times 10^9 FLOPS}{0{,}9 \times 10^9 FLOPS} = 7{,}5$$

El consumo energético de dicho procesador bajo actividad está especificado en 1.5W [17]. Las métricas que se tomaron en cuenta al comparar dichos algoritmos fueron:

- Tiempo de procesamiento: tiempo total que requirieron los nodos espaciales en computar las rutas utilizadas durante la simulación. Para la versión distribuida, el tiempo utilizado fue multiplicado por el factor de procesamiento. Si bien la versión centralizada de CGR está ideada para realizar todo el procesamiento en tierra, es útil tener una referencia de tiempo *vs.* cantidad de rutas computadas.
- Espacio utilizado: tamaño estimado para transmitir y almacenar el conjunto de rutas. La estimación fue basada en la cantidad de rutas computadas por nodos espaciales, la longitud promedio de las mismas, y el espacio estimado para identificar un contacto (salto).
- Energía utilizada: estimador para el total de energía consumida por los nodos espaciales. En el caso de la versión distribuida, el estimador fue

[4] El código puede encontrarse en `https://bitbucket.org/lcd-unc-ar/dtnsim/branch/centralized-model`

[5] `https://bitbucket.org/gasparinielias/dtnsim-scripts/src/master/notebooks/centralized-vs-distributed.ipynb`

basado en el consumo energético del procesador mencionado anteriormente. Para la versión centralizada, se consideró el consumo energético de las antenas transmisoras, ya que estas representan el gasto principal al repartir las rutas computadas en tierra hacia los nodos en la red.

6.1. Tiempo de cómputo

En las figuras 7 y 8 se pueden ver las comparaciones de tiempo utilizado por los algoritmos para el cómputo de rutas.

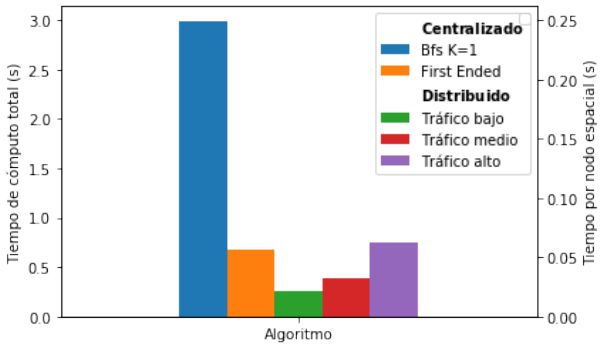


Figura 7: Tiempo total de cómputo de rutas para el escenario *walker*

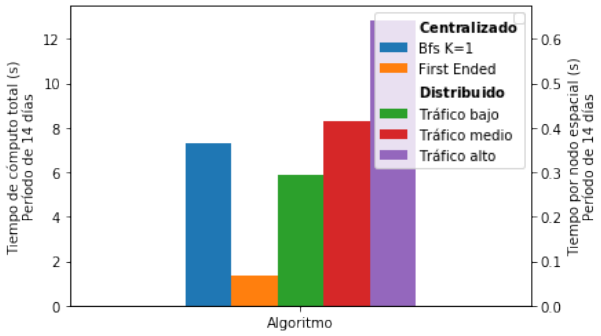


Figura 8: Tiempo total de cómputo de rutas para el escenario *Lunar Ring Road*

En primer lugar, la diferencia entre los dos algoritmos centralizados es clara: la aproximación por fuerza bruta es sustancialmente menos eficiente que la heurística *First ended*, a pesar de que ambas proveen la mejor ruta para cada par de nodos en todo instante de tiempo.

En segundo lugar, el tiempo de procesamiento por nodo para la versión distribuida es 10 veces mayor en el escenario lunar. Esto se debe principalmente a que el plan de contactos se extiende por 2 semanas, lo que amplía el tráfico de telemetría y las ejecuciones del algoritmo por nodo. Aún así, esto no ocurre con el tiempo por nodo de los algoritmos centralizados, los cuales se mantienen relativamente similares. Conduciendo pruebas piloto, se llegó a la conclusión de que este costo está parcialmente asociado a la cantidad promedio de rutas computadas por nodo. Como ejemplo, para el algoritmo *first ended*, estas cantidades son 7300 y 6900 rutas por nodo para los escenarios lunar y *walker* respectivamente, mientras que los tiempos de cómputo por nodo son 66.7ms y 56.9ms. Que el tiempo de cómputo acumulado sea el doble para el escenario lunar, se explica con el hecho de que se considera casi el doble de nodos espaciales (12 nodos espaciales en el escenario *walker*, 20 en el lunar) y que el tiempo promedio por nodo es similar. Un detalle adicional aquí es que, aunque el algoritmo distribuido tenga un uso de CPU de 0.6s en un período de 2 semanas (lunar) o un uso de 0.06s en un período de 1 día (*walker*) y pareciera ser poco para un único satélite, en realidad puede convertirse en un problema si se tiene en cuenta que esto es tiempo dedicado exclusivamente para enrutar paquetes de telemetría.

En tercer lugar, debemos destacar la diferencia entre el tráfico bajo y alto. Un aumento del 200 % en la generación de paquetes resulta en un incremento del tiempo insumido en un 200 % para el escenario *walker* y de 115 % para el lunar. Además, este costo para la versión distribuida se podría ver comprometido si el tráfico de la red fuese sustancialmente mayor que N paquetes por hora (tráfico alto de telemetría en una red de N nodos). Más aún, la versión distribuida se vió favorecida con una reutilización de rutas, dado que el tráfico de telemetría fue siempre dirigido hacia un mismo nodo central. Paquetes que pueden ser transmitidos por una ruta ya calculada no inician una nueva búsqueda de rutas. En ambos escenarios, la

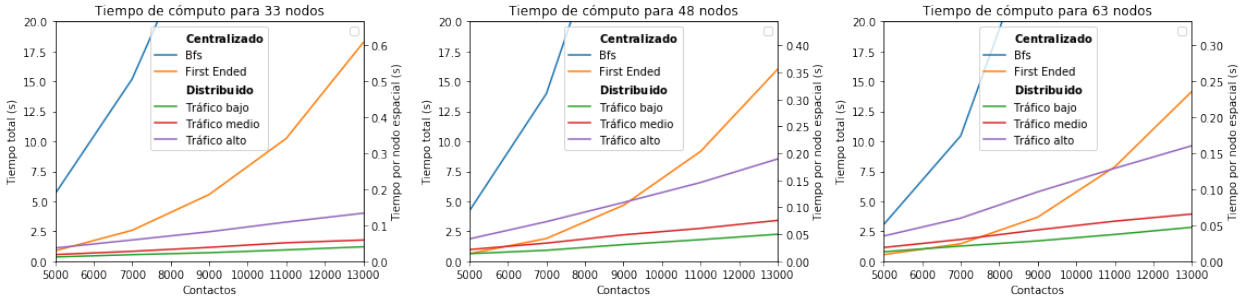


Figura 9: Tiempos de cómputo para escenarios aleatorios set 1

cantidad de paquetes que se enrutaron por rutas encontradas previamente fue superior al 25 % en el caso de tráfico alto. Esto significa que una generación de paquetes destinados hacia otros nodos produciría un aumento directo en el tiempo de enrutamiento, debido a la necesidad de obtener rutas completamente nuevas. Adicionalmente, la diferencia de tiempo entre ambos algoritmos se vería más pronunciada si la tarea centralizada se realiza en paralelo. Dicho paralelismo es posible debido a que la búsqueda por cada nodo de origen es independiente del resto.

En la figura 9, correspondiente a los escenarios aleatorios del set 1, podemos ver cómo el incremento en la cantidad de contactos impacta en los tiempos de ejecución. Es notable la curva de aumento para el caso centralizado, debido a la cantidad de ejecuciones que debe llevar a cabo, mientras que las del distribuido pueden ser aproximadas con funciones lineales. Ahora, es entendible el aumento de tiempo acumulado para la versión distribuida a medida que se incrementa la cantidad de nodos, ya que se inserta tráfico en la red. Sin embargo, el costo por nodo se mantiene bastante estable (entre 0.14s y 0.20s) (mayor costo total, pero mayor cantidad de unidades de procesamiento). No es el caso para la versión centralizada, que tiende a disminuir el tiempo utilizado por nodo mientras más grande es la red. La hipótesis aquí es que, si se mantiene la cantidad de contactos, disminuye la densidad del grafo y, por ende, la cantidad de rutas existentes. De hecho, la disminución de rutas por nodo en los casos centralizados desde 33 nodos a 63 es superior al 25 %.

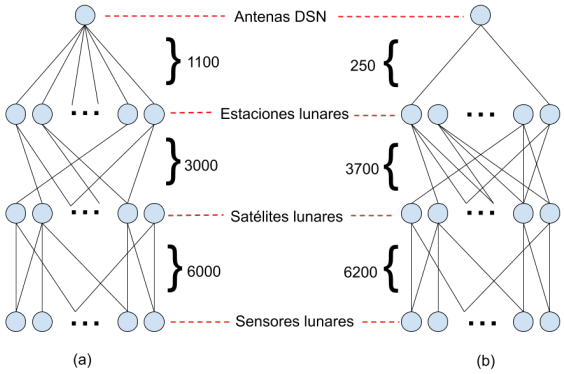


Figura 10: Esquema de distribución de contactos entre los niveles de jerarquía. (a) Distribución generada aleatoriamente, proporcional a la cantidad de nodos por nivel. (b) Distribución con sesgo hacia algún/os nivel/es en particular, correspondiente al escenario lunar.

Debemos resaltar en este punto que esta hipótesis no parece validarse con los escenarios *walker-lunar*. Recordemos que ambos rondan los 10.000 contactos y la diferencia de nodos es de 122 a 23, mientras que sus tiempos de cómputo por nodo son similares. Esto es porque la densidad del grafo original no es la única cualidad que determina la cantidad de rutas existentes en el grafo de contactos. No ahondaremos en los factores que influyen en la variación de la topología del grafo de contactos, pero dejamos dos observaciones para un posible análisis más en profundidad en trabajos futuros.

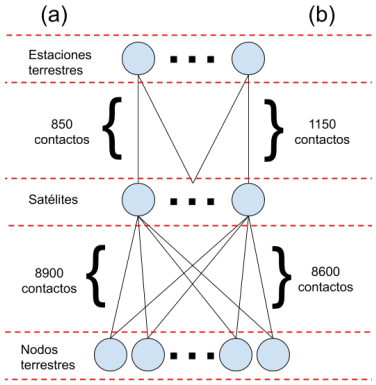


Figura 11: Esquema de distribución de contactos entre los niveles de jerarquía. (a) Escenario aleatorio semejante al *Walker*. Si bien no hay una diferencia sustancial, este caso es más proporcional a la cantidad de nodos por nivel. (b) contactos en el escenario *walker*.

Por un lado, la duración promedio por día de los contactos del escenario lunar es notablemente menor (solo un 30 %) que el resto de los escenarios. Esto afecta directamente a la definición de ruta planteada en la sección 2. A mayor duración de los contactos, mayor probabilidad de formar un tramo de ruta con dos contactos (si los contactos tuviesen duración infinita, la única condición que habría que verificar es que el destino del contacto actual coincida con el origen del contacto siguiente).

Por otro, la distribución de los contactos de ambos escenarios entre los niveles jerárquicos no es completamente aleatoria ni equitativa. En el escenario lunar, por ejemplo, esta distribución es de 254, 3688 y 6256. Para comparar este aspecto, se creó un nuevo escenario aleatorio siguiendo la misma cantidad de nodos, contactos y jerarquías que en el escenario lunar, con lo que se obtuvieron niveles de 1100, 3000 y 6000 contactos. Esta diferencia puede verse esquemáticamente en la figura 10. Una distribución más proporcional, como es el caso (a), debería tender a generar mayor cantidad de rutas, siendo que las posibles combinaciones entre contactos crecen de manera multiplicativa. La misma ilustración puede verse para el caso *walker*, emparejado con un escenario aleatorio semejante, en la figura 11.

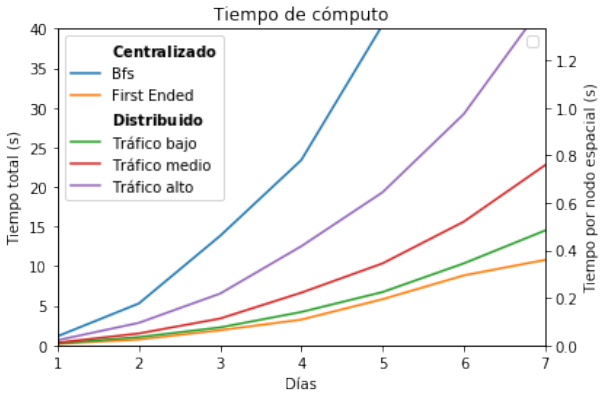


Figura 12: Tiempo de cómputo *vs.* duración del plan de contactos para los escenarios aleatorios set 2.

Para los escenarios aleatorios del set 2, se muestran los resultados de tiempo en la fig. 12. Siendo que se mantiene la cantidad de contactos promedio por día, la extensión de la simulación afecta tanto a los algoritmos distribuidos como a los centralizados. Mientras más se extiende la simulación, crece la necesidad de rutas para los nuevos instantes de tiempo. La versión distribuida satisfará esta necesidad en demanda, mientras que la centralizada lo hará precomputando las nuevas rutas en Tierra. En este gráfico, las curvas de crecimiento para la versión distribuida no toman un aspecto de función lineal como en los escenarios aleatorios set 1, ya que no solo se incrementa la cantidad de contactos a medida que crece el tiempo de simulación, complejizando el grafo de contactos, sino que también aumenta la cantidad de paquetes enrutados. Sin embargo, si los valores se promedian por día de simulación, se obtiene nuevamente una gráfica aproximadamente lineal.

6.2. Espacio utilizado

Como se mencionó en la sección 4.1, los algoritmos centralizados computan rutas preventivamente desde y hacia todos los nodos, por lo que el tiempo de procesamiento es independiente del tráfico en la

red. Esto lleva a que la cantidad de rutas generadas sea muy superior a las necesarias. Incluimos este dato en las tablas 1 y 2, junto a la longitud promedio de las rutas calculadas y una estimación de espacio para la totalidad del conjunto.

Algoritmo	Bfs	First Ended	Tráfico alto	Tráfico medio	Tráfico bajo
Paquete enviados	2928	2928	2928	1464	976
Longitud de ruta promedio	3.85	4.63	3.27	3.32	3.31
Rutas totales	622437	840836	1939	1223	891
Espacio (MB)	9.42	15.3	0.02	0.02	0.01
Rutas filtradas	6846	7351	-	-	-
Espacio (MB)	0.1	0.13	-	-	-

Tabla 1: Escenario *walker*: cantidad de rutas computadas

Algoritmo	Bfs	First Ended	Tráfico alto	Tráfico medio	Tráfico bajo
Paquete enviados	7728	7728	7728	3864	2576
Longitud de ruta promedio	3.72	4.0	3.74	3.72	3.71
Rutas totales	152442	168710	3763	2573	1761
Espacio (MB)	2.23	2.65	0.06	0.04	0.03
Rutas filtradas	5479	5655	-	-	-
Espacio (MB)	0.08	0.09	-	-	-

Tabla 2: Escenario *Lunar Ring Road*: cantidad de rutas computadas

Es importante notar la gran diferencia (3 órdenes de magnitud) en la cantidad de rutas computadas por todos los nodos entre la versión centralizada y distribuida. Si bien en el caso centralizado este volumen es una carga extra para el tráfico en la red, en la práctica se lo puede mantener al mínimo si se realiza un filtrado de rutas de acuerdo al tráfico esperado, los potenciales nodos de destino, y/o a la misión espacial establecida. En este caso, para el tráfico de telemetría, consideramos el filtrado de rutas como la eliminación de toda ruta que no tenga como destino el CCM, lo que resultó en una reducción mayor al 95 %.

Con respecto a la estimación de espacio, asumimos que cada plan de contactos tiene una forma de ser identificado, así como los contactos que lo componen. Entonces, transmitir una ruta puede llevarse a cabo enviando la sucesión de los identificadores de contactos que la conforman más el identificador del plan de contactos al que pertenecen. Esta suposición es razonable, ya que (a) el plan de contactos es necesario en cada nodo, independientemente de la forma de enrutamiento; y (b) dadas las posibles repeticiones en las rutas, enviar todos los datos que definen cada contacto es ineficiente. Por esto, el cálculo para obtener la cantidad de *bytes* que ocuparía un conjunto de R rutas fue:

$$(R * longitud\_de\_ruta\_promedio + 1) * 4B$$

A este valor se le agregó el 3 %, correspondiente al *overhead* (sobrecarga) producido por las capas inferiores de transmisión, estimado en [18].

En las tablas 1 y 2 también se puede ver que los algoritmos centralizados no encuentran la misma cantidad de rutas. La causa de esto es la naturaleza misma de búsqueda de cada uno: ante dos posibles rutas con igual métrica, la elección es arbitraria y depende del algoritmo. Más aún, una bifurcación temprana en las rutas encontradas lleva a que las modificaciones al plan de contactos sean distintas y, por lo tanto, las rutas subsiguientes. A pesar de ello, mostramos en la subsección 6.4 que ambos algoritmos coinciden en todo momento con respecto a la métrica "tiempo de arribo". Recordamos aquí que el algoritmo de fuerza bruta está severamente limitado por el parámetro K (cantidad máxima de rutas hacia un destino e instante de tiempo particular). Es por este motivo que el BFS encuentra una cantidad similar (incluso menor) de rutas que el algoritmo *first ended*. Pruebas con K=2 y K=3 llevan a duplicar y triplicar, respectivamente, esta cantidad.

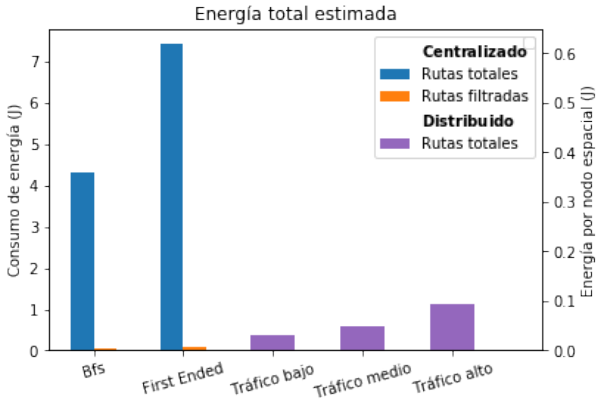


Figura 13: Escenario *walker*: Consumo de energía debido a: (a) procesadores durante el cómputo de rutas en el caso distribuido, (b) antenas transmisoras satelitales durante el envío de las rutas encontradas de manera centralizada

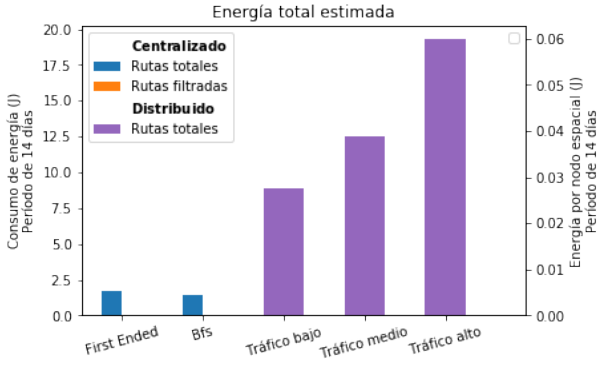


Figura 14: Escenario *Lunar ring road*: Consumo de energía debido a: (a) procesadores durante el cómputo de rutas en el caso distribuido, (b) antenas transmisoras satelitales durante el envío de las rutas encontradas de manera centralizada

6.3. Estimación de energía

A continuación se muestra el impacto de la cantidad de rutas sobre la energía total utilizada. Para la versión centralizada, se tomó como referencia el transmisor *X-Band Transmitter* [19], con un consumo de 12W durante transmisión y una capacidad de hasta 150Mbps. De acuerdo a la estimación de tamaño de las rutas, se calculó el consumo esperado por los transmisores en los nodos espaciales. Se tomó en cuenta que para transmitir las rutas correspondientes hasta un nodo ubicado en un nivel L, es necesario realizar $L-2$ transmisiones por nodos espaciales, siendo que el nivel 1 son antenas terrestres en todos los escenarios.

En las figuras 13 y 14, se hace visible la ganancia en la reducción de rutas. Si se espera que el tráfico tenga como destino final un conjunto reducido de nodos, es de importancia filtrar las rutas adecuadas. Una planificación estricta y bien informada permite mantener la sobrecarga en la red y el costo de transmisión al mínimo. Este efecto se hace más visible en redes grandes, puesto que los posibles pares nodo-destino crecen de manera cuadrática con respecto a la cantidad de nodos en el grafo.

Se observa también que la diferencia entre las versiones centralizadas difiere con respecto a las figuras mostradas en la sección de timepo: el algoritmo *first ended* tiene mayor costo que el BFS, pues la métrica de energía para estos algoritmos está puramente basada en la cantidad y longitud de las rutas que cada uno encuentra. Por su parte, los algoritmos distribuidos están linealmente relacionados con la métrica de tiempo utilizado en el cómputo de rutas.

Para los escenarios aleatorios del set 1 (figura 16), se omitieron las líneas correspondientes a la versión centralizada sin filtrado de rutas dada la excesiva curva de crecimiento, pasando los 50J para 63 nodos y 13000 contactos. No se continuaron las simulaciones con el algoritmo de BFS por el tiempo que conlleva. En tanto los gráficos parecerían mostrar una ganancia relativa muy significativa en términos de energía (de alrededor del 500 %) por el uso de algún algoritmo centralizado, en números concretos, esta diferencia es menor a 0.3J por nodo en todos los casos. No obstante, la curva de gasto de energía del enfoque centralizado tiene una menor pendiente que el resto. Esto podría indicar que la transmisión de datos tiene mejor eficiencia, y por lo tanto escalabilidad, que los procesadores espaciales en término de rutas propor-

cionadas por unidad de energía.

Por último, se muestra en la figura 15 la misma métrica para los escenarios aleatorios del set 2. En ella se puede ver la evolución de la energía utilizada a medida que se incrementa la proyección del plan de contactos. Nuevamente, por la baja cantidad de nodos y la ayuda con el filtrado de rutas, la energía consumida por los algoritmos centralizados se mantiene al mínimo.

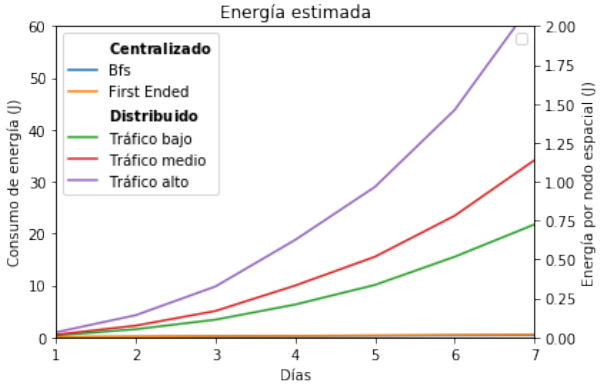


Figura 15: Energía estimada para los Escenarios aleatorios set 2

6.4. Verificación de rendimiento

Para asegurarnos de que el algoritmo centralizado se comporta de la manera esperada, se agregaron métricas que respectan a los tiempo de entrega de los paquetes para los escenarios *walker* y *Lunar* (figuras 17 y 18 respectivamente). Los gráficos de línea a la izquierda muestran el tiempo estimado promedio de arribo de los mensajes al ser enrutados por el nodo emisor. El promedio se realizó sobre todos los mensajes que fueron generados en el mismo instante de tiempo.

Como se ve, las líneas se superponen practicamente en todo momento. Esto indica una gran similitud en la métrica "tiempo de arribo"de las rutas elegidas por todos los algoritmos de enrutamiento, por más que las rutas sean distintas. Vale notar que, aunque las variaciones de tráfico medio y bajo envían menor cantidad de paquetes, las curvas siguen la misma trayectoria que las otras, quizás con menor "detalle"(puntos en la gráfica). Aún así, esta es una métrica tomada al momento de emisión del paquete, sin tener en cuenta los mensajes ya destinados para la misma ruta por el nodo misno y los nodos subsiguientes. Por ello es necesario corroborar que las demoras de entrega en el CCM, nodo destino, sean similares entre la versión distribuida y la centralizada.

Se decidió no mostrar gráficos de línea con la información "delay *vs* tiempo de arribo"por dos motivos: a) Los mensajes correspondientes a un tiempo de emisión llegan en tiempos muy dispares, dependiendo del nodo de partida. Esto hace que se generen muchos picos en la gráfica, correspondientes a los distintos ciclos de generación, haciéndola difícil de interpretar. b) Siendo que los algoritmos pueden variar en las rutas encontradas y las que eligen para enrutar, un mismo paquete en simulaciones diferentes puede no llegar en tiempos iguales. Los histogramas del lado derecho muestran las cantidades de paquetes que han sido entregados en un determinado rango de *delay* (demora). Si bien los rangos de demora parecen ser pocos y muy abarcativos, se hicieron pruebas con 20, 40 y 80 rangos sin obtener una diferencia en los resultados: el algoritmo distribuido se mantiene al mismo nivel que los dos centralizados. Esto es, para una cierta tolerancia de variación de demora, los algoritmos logran enrutar paquetes a la vez que mantienen un tiempo de entrega similar.

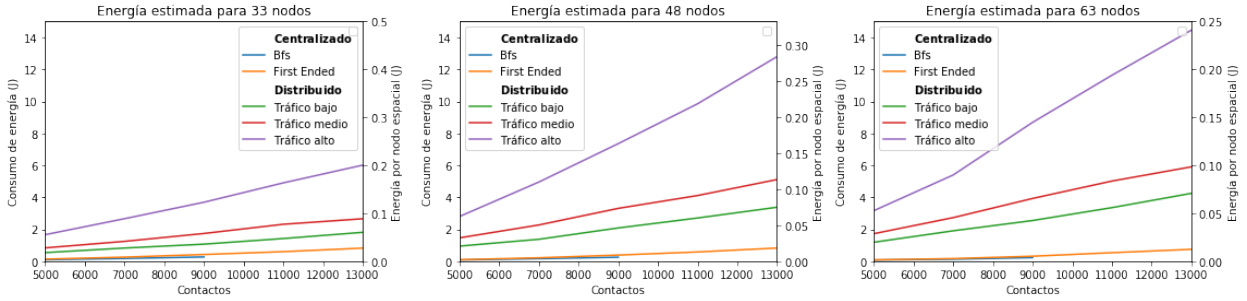


Figura 16: Energía estimada para los escenarios aleatorios set 1. Se consideran solo las rutas filtradas para los casos centralizados.

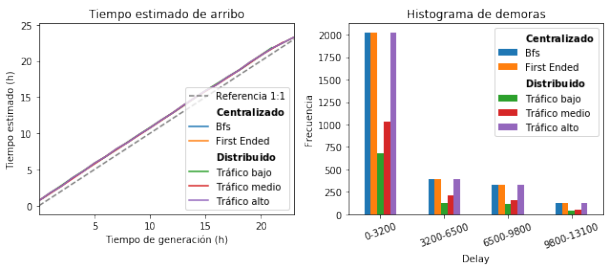


Figura 17: Verificación de *delay* sobre el escenario *walker*. A la izquierda, se visualiza que todos los algoritmos seleccionan rutas con tiempos estimados de arribo muy similares. A la derecha, las frecuencias de llegada para distintos intervalos de demora. El tráfico medio y bajo difieren en las cantidades por enviar menor cantidad total de paquetes.

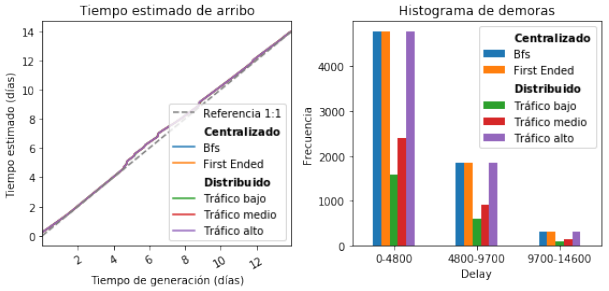


Figura 18: Verificación de *delay* sobre el escenario *Lunar*. A la izquierda, se visualiza que todos los algoritmos seleccionan rutas con tiempos estimados de arribo muy similares. A la derecha, las frecuencias de llegada para distintos intervalos de demora. El tráfico medio y bajo difieren en las cantidades por enviar menor cantidad total de paquetes.

7. Conclusión y trabajos futuros

En este trabajo se abordó el problema de enrutamiento de paquetes en redes DTN-predecibles, particularmente estudiando el costo asociado a la ejecución de rutinas en nodos con bajos recursos de procesamiento y energía. Se comparó el actual enfoque distribuido del protocolo CGR, con búsqueda de rutas bajo demanda de tráfico y heurística *first depleted*, con uno centralizado propuesto en este trabajo, en donde el cómputo de rutas ocurre en Tierra. Para este último, se introdujeron dos algoritmos para realizar la búsqueda de rutas, de manera tal que se provea a todos los nodos con las rutas necesarias al momento de enrutamiento y así evitar la ejecución de procedimientos costosos en órbita. Dichos algoritmos, si bien son adaptaciones de otros ya existentes, constituyen la primera aproximación a una visión centralizada de CGR. Los resultados se obtuvieron por medio de simulaciones para un tráfico de telemetría en redes generadas tanto con parámetros orbitales reales como de manera aleatoria.

Una primera exploración confirmó que la diferencia de consumo de tiempo y energía entre la versión distribuida y centralizada está ligada al tráfico de la red, así como al tamaño del plan de contactos. Naturalmente, estos son factores importantes que impacta en la complejidad de tiempo y la cantidad de veces que deben ejecutarse las rutinas de búsqueda en los nodos espaciales. Adicionalmente, detectamos que la distribución de contactos entre los nodos afecta sustancialmente a la topología del grafo de contactos y, por lo tanto, a las métricas estudiadas. De hecho, el estudio de la morfología del grafo de contactos se identifica como un interesante campo de estudio en el área. En este marco, propusimos una nueva metodología para comparar justamente ambas aproximaciones. La misma consta de llevar los costos principales de cada enfoque (tiempo de cómputo en los nodos espaciales para la versión distribuida y tamaño de las rutas computadas en Tierra para la centralizada) a términos de gasto de energía. De esta manera, se pudo estimar la conveniencia de uso de una forma de enrutamiento sobre la otra dados los algoritmos de búsqueda de ruta, las especificaciones de consumo de *hardware* y el tráfico esperado.

Los resultados de simulación mostraron que el enfoque centralizado propuesto en este trabajo es superior al distribuido cuando se puede evitar transmitir a los nodos rutas innecesarias, como ser rutas cuyo destino final sea nodos en donde no se espera llegada de paquetes. Este filtrado de rutas, obtenidas de manera centralizada, resultó fundamental para mantener al mínimo el espacio de almacenamiento, la sobrecarga en la red y la energía utilizada por las antenas transmisoras. Los resultados (resumidos en la fig. 19) indican una ganancia en energía de hasta 0.2J por día por nodo a favor de los algoritmos centralizados. Aunque en algunos casos prácticos esta diferencia no resulte significativa en términos cuantitativos, resaltamos otros aspectos relevantes que favorecen la versión centralizada de CGR aquí discutida:

(a) El tiempo utilizado de manera centralizada no representa un costo real para los satélites, ya que el procesamiento ocurre en tierra. En consecuencia, si bien el cómputo centralizado mostró escalar peor que el distribuído, el enfoque centralizado resulta particularmente atractivo para redes pequeñas (de hasta 100 nodos y 10.000 contactos, lo que abarca la mayoría de las redes espaciales actuales).

(b) La disponibilidad de mayor poder de cómputo en tierra y la posibilidad de paralelismo durante la búsqueda de rutas perfilan al esquema centralizado como un buen candidato para futuras redes de tamaño intermedio (de hasta 300 nodos y 50.000 contactos). Esto se suma con el hecho de que los gráficos de energía con filtrado de rutas proyectan una mejor escalabilidad del enfoque centralizado. En consecuencia, el costo de transmitir las rutas hasta los satélites terminó siendo menor que el de computarlas en órbita en todos los escenarios analizados.

(c) El encabezado *source routing*, utilizado en este trabajo, favorece al esquema distribuído (la ruta sólo se computa en el origen). Sin embargo, en la práctica, CGR re-computa la ruta en todos los nodos, lo que empeoraría las métricas aquí discutidas. Esto dió una ventaja notable al algoritmo distribuido tanto en tiempo como energía.

Es por estas razones que concluimos que el modelo centralizado de CGR propuesto en este trabajo

no solo es factible, sino que es favorable en el corto y mediano plazo.

	Tiempo p/nodo (s)	Espacio total (MB)		Energía p/día p/nodo (mJ)	
		c/filtrado	s/filtrado	c/filtrado	s/filtrado
Walker (10k.cont/ 122nod/1día)	C: 0.05 (0*) D: 0.05	C: 0.13 D: –	C: 15.3 D: 0.02	C: 6 D: –	C: 600 D: 100
Lunar (10k.cont/ 23nod/2sem)	C: 0.05 (0*) D: 0.6	C: 0.09 D: –	C: 2.65 D: 0.06	C: 0.2 D: –	C: 6 D: 59
Aleat.S1 48nod.11k.cont (varía nod/cont)	C: 0.20 (0*) D: 0.15	---	---	C: 13 D: –	C: 600 D: 200
Aleat.S2 5 días (varía período)	C: 0.2 (0*) D: 0.6	---	---	C: 2 D: –	C: 62 D: 190

C: Centralizado - first ended
D: Distribuido - first depleted - tráfico alto
(*) Tiempo centralizado no es costo en los satélites, sino en Tierra.

Figura 19: Resumen de resultados.

Queda como trabajo pendiente explorar cómo otras distribuciones de tráfico afectan a ambas formas de enrutamiento. Los problemas de congestión y de saturación de la capacidad de las rutas, dejados de lado en nuestro análisis, deberán ser abordados con una predicción más acertada en el caso centralizado. Será necesario para situaciones de uso pico de la red explorar algoritmos centralizados más eficientes que el BFS propuesto en este trabajo que permitan la obtención de más de una ruta por cada instante de tiempo y así aprovechar la capacidad de la red. Desde luego que esto implica aumentar el volumen de rutas transmitidas hacia los satélites. Por otro lado, no descartamos la posibilidad de un enfoque híbrido, en el cual se computen en Tierra las rutas con mayor probabilidad de uso, mientras que se toma una aproximación distribuida en caso de congestión o tráfico inesperado. Con respecto a la morfología del grafo de contactos, deberá investigarse la influencia de la distribución de contactos entre los nodos en la cantidad de rutas generadas. Por último, la escalabilidad de las dos versiones de enrutamiento sigue bajo estudio para redes grandes, con más de 1000 nodos, estimadas a ponerse en práctica en un futuro no lejano.

Referencias

[1] K. Scott and S. Burleigh, "Bundle protocol specification," Internet Requests for Comments, RFC Editor, RFC 5050, November 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5050.txt

[2] 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, pp. 1980 – 1997, 11 2011.

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

[4] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and wait: An efficient routing scheme for intermittently connected mobile networks," *Proceedings of ACM SIGCOMM workshop on Delay-tolerant networking*, 09 0002.

[5] 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: Springer Berlin Heidelberg, 2004, pp. 239–254.

[6] 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)*, 2018, pp. 1–6.

[7] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," vol. 34, 10 2004, pp. 145–158.

[8] S. Burleigh, "Contact graph routing, IETF-Draft draft-burleigh-dtnrg-cgr-01," Jul 2010.

[9] J. A. Fraire and J. M. Finochietto, "Design challenges in contact plans for disruption-tolerant satellite networks," *IEEE Communications Magazine*, vol. 53, no. 5, pp. 163–169, 2015.

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

[11] J. Y. Yen and J. Y. YENt, "Finding the k shortest loopless paths in a network," 2007.

[12] S. Biswas, B. Alam, and M. Doja, "Generalization of dijkstra's algorithm for extraction of shortest paths in directed multigraphs," *Journal of Computer Science*, vol. 9, pp. 377–382, 01 2013.

[13] 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)*, 2017, pp. 1–7.

[14] M. Feldmann, J. Fraire, and F. Walter, "Tracking lunar ring road communication," 05 2018, pp. 1–7.

[15] J. A. Fraire, P. Madoery, F. Raverta, J. M. Finochietto, and R. Velazco, "Dtnsim: Bridging the gap between simulation and implementation of space-terrestrial dtns," in *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, 2017, pp. 120–123.

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

[17] "Proton 200k specifications," https://www.spacemicro.com/assets/datasheets/digital/slices/proton200k-L.pdf, 2015.

[18] C. C. for Space Data Systems, "Schedule-Aware Bundle Routing: recommended standard." https://public.ccsds.org/Pubs/734x3b1.pdf, 2019.

[19] "Endurosat X-Band Transmitter," https://www.endurosat.com/cubesat-store/cubesat-communication-modules/x-band-transmitter/, 2020.

8. Apéndice

8.1. Dijkstra en multigrafo

Este algoritmo no presenta mayores diferencias con el algoritmo de búsqueda de camino de costo minimal de Dijkstra en su implementación con una cola de prioridad para seleccionar el siguiente nodo a visitar. Aún así, aclaramos a continuación algunos puntos en particular. Por un lado, la adaptación para multigrafo se puede ver en el registro del contacto predecesor para cada nodo, en vez de el nodo predecesor. Dado que múltiples aristas pueden existir entre dos nodos, reconstruir el camino requerirá determinar cuál de ellas fue la utilizada. En la línea 1 se crea esta asociación nodo-arista, y solo se utiliza cuando se encuentra un mejor camino hacia un nodo (línea 16) y para la reconstrucción de rutas (línea 18). Solo por claridad, mencionamos que el costo hacia cada otro nodo se almacena en "Tiempos_entrega"(línea 2), inicialmente con costo 0 para el nodo de origen e infinito para el resto.

Por otro lado, y como ya se ha mencionado, la definición de "arista."[en] este grafo depende del tiempo. Esto se refleja en las líneas 10-11, donde se analizan las aristas partiendo del nodo en visita. Se deben cumplir las condiciones notadas en la sección 3 para que una sucesión de contactos sea considerada una ruta válida. Esta selección de arista se puede optimizar con un precómputo básico en donde se organicen los contactos separándolos por nodo de origen. De la misma forma, el costo de una arista no es fijo y se determina en la línea 14, según la definición recursiva de la sección 2.

Algoritmo 3: Dijkstra en multigrafo

Entrada: Plan de contactos CP, nodo raíz O, conjunto de nodos en la red $\mathcal{S}$
Resultado: Lista de rutas R, árbol de costo minimal

```
1  Contactos_predecesores ← {} ;
   Tiempos_entrega ← {}
2  Nodos_visitados ← {}
3  PQ ← {O}  // Cola de prioridad con
   nodos ordenados por mejor tiempo de
   entrega
4
5  while PQ.tiene_elementos() do
6      Nodo ← PQ.pop()
7      if Nodo ∈ Nodos_visitados then
8          continue
9      Nodos_visitados.agregar(Nodo)
10     foreach contacto C ∈ CP tal que
        source(C) = Nodo do
11         if end(C) ≤ Tiempos_entrega[Nodo]
            then
12             continue
13         Vecino ← dest(C)
14         Costo_hacia_vecino ←
            max(Tiempos_entrega[Nodo],
            start(C)) + delay(C) ; if
            Costo_hacia_vecino <
            Tiempos_entrega[Vecino] then
15             Tiempos_entrega[Vecino] ←
                Costo_hacia_vecino
16             Contactos_predecesores[Vecino] ←
                C
17             PQ.push(Vecino)
18 R ← Construir_rutas(Contactos_predecesores)
19 return R
```

8.2. BFS

La búsqueda de rutas en este algoritmo se da a modo de fuerza bruta, explorando las posibilidades a modo de BFS, es decir, primero las que tienen menor cantidad de saltos. Recordemos que la búsqueda está limitada por los parámetros K (cantidad máxima de rutas hacia un destino en particular) y M (cantidad máxima de saltos en una ruta). Partiendo de una ruta inicial, desde el nodo de origen (línea 7), se comienzan a analizar las rutas encoladas. Para cada una, se consideran todos los contactos tales que puedan extenderla (que cumplan la condición de ruta válida, línea 14). La manera de mantener las mejores K rutas hacia un destino en particular se realiza con una cola de prioridad (línea 2). Cuando una nueva ruta es encontrada hacia dicho destino, se la compara con la peor de la cola (tiempo constante) y se la inserta en caso de ser mejor. Notar que si dos rutas R_1 y R_2 cumplen $tiempo\_de\_entrega(R_1) < tiempo\_de\_entrega(R_2)$ y a su vez R_2 puede ser extendida con el contacto C, es decir, $tiempo\_de\_entrega(R_2) < end(C)$, entonces R_1 también puede ser extendida por el contacto C. Esto nos dice que no es necesario explorar una ruta si ya se tienen K rutas mejores hacia el mismo destino. Por ello, la línea 17 considera el único caso en que se debe eliminar una ruta de la cola de prioridad: se halla una mejor ruta que la peor de las K previamente encontradas y no haya espacio para almacenarla.

Algoritmo 4: Búsqueda BFS

Entrada: K, M, plan de contactos CP, nodo raíz O, conjunto de nodos en la red $\mathcal{S}$

Resultado: Lista de rutas por nodo de destino

```
// Inicialización
1  foreach nodo de destino s ∈ S do
2  |  rutas_por_destino[s] ← {}
3  contacto_virtual = C^{0,∞,∞}_{O,O,0}
4  ruta_virtual =
     ruta_a_partir_de_contacto(contacto_virtual)
   // Agregar ruta con solo el contacto
      virtual
5  rutas_para_explorar ← {ruta_virtual}
   // Buscar rutas
6  while rutas_para_explorar no vacía do
7  |  R ← rutas_para_explorar.pop()
8  |  D ← destino(R)
9  |  if size(rutas_por_destino(D)) = K ∧ ∄r ∈
   |    rutas_por_destino[D] tal que r es peor que
   |    R then
10 |  |  continue // No analizar ruta si es
   |  |     peor que las K ya encontradas
11 |  foreach contacto C ∈ CP tal que
   |    source(C) = D ∧
   |    tiempo_de_arribo(R) < end(C) do
12 |  |  nueva_ruta ←
   |  |    R.extender_con_contacto(C)
13 |  |  d ← destino_de(nueva_ruta)
14 |  |  if ∃r ∈ rutas_por_destino[d] tal que r
   |  |    es peor que nueva_ruta ∧
   |  |    size(rutas_por_destino[d]) = K then
15 |  |  |  rutas_por_destino[d].eliminar(r)
16 |  |  if size(rutas_por_destino[d]) < K then
17 |  |  |  rutas_por_destino[d].agregar(nueva_ruta)
   |  |  |
18 |  |  |  if nueva_ruta.cantidad_de_saltos <
   |  |  |    M then
19 |  |  |  |  rutas_para_explorar.agregar(nueva_ruta)
   |  |  |  |
20 return rutas_por_destino
```

8.3. Contribución a DTNsim

El principal aporte en código para el simulador fue la implementación de un nuevo módulo de enrutamiento: RoutingCgrModelCentralized. Esta clase abarca el funcionamiento de enrutamiento centralizado en la capa de red. Mantiene la misma interfaz que los módulos de enrutamiento distribuidos, pero realizando el cómputo de rutas únicamente al ser inicializada[6]. Los parámetros soportados son: algoritmo de búsqueda de rutas (BFS, first ended) y utilización del bloque de extensión *source routing*.

Adicionalmente, se optimizaron los siguientes módulos: ContactPlan y CgrModelRev17. En el primero, las operaciones de obtener contactos filtrados por nodo de origen, obtener rango para un contacto y obtener contacto por ID fueron precomputadas durante la inicialización, logrando una reducción de complejidad de $\mathcal{O}(C)$ a $\mathcal{O}(1)$, donde C es la cantidad total de contactos. Con respecto al segundo, al algoritmo de Dijkstra utilizado para la búsqueda de rutas se le agregó la cola de prioridad para disminuir su complejidad total de $\mathcal{O}(C^2+E)$ a $\mathcal{O}(C*\log C+E)$, donde C es la cantidad de contactos (nodos en el grafo de contactos) y E es la cantidad de aristas del grafo de contactos.

[6] Al momento, DTNsim no soporta el cambio del plan de contactos durante la simulación. Si esta funcionalidad fuera implementada, se le deberá agregar al módulo centralizado una nueva llamada al procesamiento de rutas. Esto no debería ser necesario para los módulos distribuidos, que computan rutas bajo demanda.