Experimental Evaluation of On-Board Contact-Graph Routing Solutions for Future Nano-Satellite Constellations

"Blas F. Vega"
"Agencia Espacial del Paraguay"
"Av. Defensores del Chaco, San Lorenzo-Paraguay" ; +595961409189
bvega@aep.gov.py
"Juan A. Fraire"
" CONICET - Universidad Nacional de Córdoba"
"Av. Vélez Sarsfield 2000, Córdoba, Argentina"; +5493512446010
juanfraire@unc.edu.ar
and
"Saarland University"
"Saarland Informatics Campus, Saarbrücken, Germany"

ABSTRACT

Hardware processing performance and storage capability for nanosatellites have increased notably in recent years. Unfortunately, this progress is not observed at the same pace in transmission data rate, mostly limited by available power in reduced and constrained platforms. Thus, space-to-ground data transfer becomes the operations bottleneck of most modern space applications. As channel rates are approaching the Shannon limit, alternative solutions to manage the data transmission are on the spot. Among these, networked nano-satellite constellations can cooperatively offload data to neighboring nodes via frequent inter-satellite links (ISL) opportunities in order to augment the overall volume and reduce the end-to-end data delivery delay. Nevertheless, the computation of efficient multi-hop routes needs to consider not only satellite and ground segments nodes, but a non-trivial time dynamic evolution of the system dictated by orbital dynamics. Also, in most practical cases, the forwarding decision shall happen in orbit, where satellites can timely react to local or in-transit traffic demands. In this context, it is appealing to investigate on the applicability of adequate algorithmic routing approaches running on state-of-the-art nanosatellite on-board computers. In this work, we present the first implementation of Contact Graph Routing (CGR) algorithm developed by the Jet Propulsion Laboratory (JPL, NASA) for a nanosatellite on-board computer. We describe CGR, including a Dijkstra adaptation operating at its core as well as protocol aspects depicted in CCSDS Schedule-Aware Bundle Routing (SABR) recommended standard. Based on JPL's Interplanetary Overlay Network (ION) software stack, we build a strong baseline to develop the first CGR implementation for a nano-satellites. We make our code available to the public and adapt it to the GomSpace toolchain in order to compile it for the NanoMind A712C on-board flight hardware based on a 32-bit ARM7 RISC CPU processor. Next, we evaluate its performance in terms of CPU execution time (tick counts) and memory resources for increasingly complex satellite networks. Obtained metrics serve as compelling evidence of the polynomial scalability of the approach, matching the predicted theoretical behavior. Furthermore, we are able to determine that the evaluated hardware and implementation can cope with satellite networks of more than 120 nodes and 1200 contact opportunities.

INTRODUCCION

The New Space Context

The demand of processor and memory performance has increased dramatically in recent years. To satisfy the demand, COTS (Components of The Shelf) devices have been made available and considered for use in small satellites, significantly reducing the development costs. As a result, cubesats missions can leverage significant economic and lead time advantage compared with traditional mission standards.

Embedded systems had enjoyed important innovations. Powerful processors with reduced instruction set (RISC architecture) and memories with larger capacity are now possible thanks to miniaturization in electronic components. These revolutionary changes also impacted in the size and capacity of state-of-the-art small satellites [1]. Missions otherwise impossible due the high costs and complexities are now accessible to many, including universities and start-up companies looking for novel business opportunities. We are indeed living the so-called "democratization" of space.

As a consequence of this context, cubesat mission designers can profit from a wide-range of immediately available resources ranging from satellite components, subsystems and software elements both for flight and ground segments. Boosted by scale production of standardized mechanical and electrical interfaces, satellite integration can thus be accomplished at unprecedented speed. This is reflected in the number of cubesats launched in recent years [2, 3], and in the notable success achieved by educational development programs (e.g., Birds Program of The University of Kyutech, Kyushu, Japan). Thanks to this, several countries of emerging economies have been able to afford, access and launch their first satellite [4].

The paradigm shift also extends to the satellite mission design and integration processes. Originally, every detail had to be documented and each component tested via destructive actions in order to provide the required reliability confidence. In contrast, the "new space" approach selectively relaxes the security steps in favor of a reduced mission cost and time-frame.

Despite these advances, some degree of reliability needs to be imposed to COTS elements conceived to operate under the relatively "calm" environment below the atmosphere. In particular, the presence of radiation, mechanical stress, and extreme temperature ranges demand for screening processes before flying the mission [5, 6]. Although a noticeable risk reduction with respect to traditional missions, innovating in networked data management systems remains a challenging objective.

Communications in the New Space

According to [7], the majority of operative satellite missions were deployed to provide communications and data relay services. Other applications such as earth observation and navigation follow next. The reason behind this lies on the increasing demand for data by an ever growing and digitalized population. The recent trend of connected objects or Internet of Things is also a major motivation for communications from space [8].

Communication missions are not only based on ground links but also on satellite-to-satellite or inter-satellite links. Popular networked missions such as Samsung's [9], Starlink and OneWeb are compelling evidence of the private sector's interest in building networked satellite constellations despite the complexity that their maintenance, handling, and coordination. These ambitious missions expect to provide world-wide internet coverage and decrease data delivery latency. However, to succeed on this objective, these constellations are built with thousands of satellites, only suitable for a few large-scale companies in the planet.

Alternative options are desirable in a truly democratized New Space context. One way of coping with the problem of reduced visibility with the satellite had been traditionally solved by means of geostationary satellites with large coverage. Nevertheless, geostationary relay systems tend to be unaffordable for cubesat missions with constrained budgets. Another possibility is to deploy several ground stations in strategic locations (i.e., near the pole in case of quasi-polar orbits). The issue of this option is that this typically involves lengthy bureaucratic or political barriers that are also not suitable for the agile vision of the new space.

A DELAY-TOLERANT PROPOSAL

A more efficient solution for the new space is possible. Instead of increasing the number of ground stations on inaccessible regions or deploying overly expensive geostationary satellites, opportunistic and best-effort inter-satellite link between satellites in the same constellation can be exploited.

Opportunistic contacts between orbiting spacecraft are by definition episodic and forbid a continuous and stable end-to-end data flow as assumed for traditional Internet protocols. To cope with this situation, Delay Tolerant Networking (a.k.a. DTN) exploits a novel data flow approach: store-carry-and-forward. Since DTN relaxes the strict end-to-end data path requirement (the one that basically demand thousands of satellites for a continuous global coverage), it is particularly appealing for sparse constellations of a few cubesats. In DTN, data can be moved from one spacecraft to the other, and remain arbitrarily long period of time in memory, until an adequate next-hop link becomes available. The fact that data is not immediately transferred to the final destination, gives this paradigm its name: delay-tolerant.

The main challenge in DTN space networks is to design and implement suitable routing algorithms to support the optimal determination of an adequate next-hop node and also, a suitable transmission window. This bi-dimensional routing problem is already more complex than routing on Internet, as it enjoys a stable topology on which the time-dimension is not present. In a time-evolving space topology, time is off the essence. In particular, it is not matter of simply finding to "whom" a give packet of data should be transferred, but also "when" this should happen. Resulting decision shall be driven by earliest delivery time or similar metrics [10].

DTN networks have received the attention of the community since early 2000s [11] Originally, it was proposed as an architecture to deploy a Solar-System Internet, namely, an interplanetary network infrastructure where signal propagation delay and disruptions due to planet occlusion are the rule and not the exception. Consequently, the topology of a DTN is better described by "contacts" instead of "links" as in traditional network graphs.

A contact is formally defined as an episode of connectivity between two nodes, on which data can be transferred between a transmitter and a receiver node. Indeed, by definition, a contact starts and ends at a given time, and is characterized by a duration equal to the time difference between start and end times. Also, contacts can have arbitrarily long signal propagation delays. In other words, signals are not expected to arrive immediately to the destination as in Internet networks. It

is interesting to note that a continuous link connectivity can still be represented by the more general contact concept, where duration is set to infinite, and the propagation delay to zero. As a result, a time-evolving topology formed by orbiting spacecraft in near-Earth of deep-space can be properly modeled and integrated with ground cabled infrastructure.

The routing of data packets allows the communication despite intermittent connectivity by pre-calculating changes in contacts between nodes, which can be imprinted int the *contact plan*. The contact plan is distributed to the nodes so that each node can know the specific time at which it can communicate with another node. Then each node must be able to resolve the next route to transmit the data packet in the network that constantly varies over time using the "Contact Graph Routing" (CGR) algorithm. Instead of having a defined path, in CGR, the data packet hop to the next node is resolved and defined locally on each node, thus, it provides the ability to adapt to network changes and data demand.

Considering future satellite missions where this algorithm can play a fundamental role, such as resource-constrained cubesats, it is worth the effort to study the performance of small computers or microprocessors. In particular, these are widely available for nano-satellites. We are interest in analyzing empirical results that could help to decide upon future missions based on delay tolerant networking.

DTN AND CGR IN A CONSTRAINED OBC

NanoMind OBC Test-bench

Software for on-board spacecrafts require strict approaches to be used in the verification process. This kind of analysis, in turn, requires the worst-case execution time (WCET) of each task to be known. In particular, the present experiment can be used as a first step in performing timing analysis on routing computations for a swarm of limited embedded on-board systems

We leverage the NanoMind A712c, with a 32-bit ARM7TDMI processor for CubeSats running as a task in FreeRTOS [12], which is available at the laboratories of "Universidad de Formación Superior", part of the educative framework at the Argentinian Space Agency (CONAE). The distinguishing features of this OBC are:
• High-performance 32-bit ARM7 RISC CPU
• FreeRTOS and eCos realtime operating systems
• Clock speed: 8-40 MHz
• 2MB Static RAM
• 4MB Data Storage (Flash Memory)
• 4MB Code Storage (Flash Memory)
• 104-pin CubeSatKit bus connector

For this hardware, a custom benchmark to evaluate the CGR subroutine of the ION (Interplanetary Overlay Network) software implementation was developed. Inspiration to define the architecture was taken from the DTN architecture in RFC 4838, DtnSim [13] and ION. The CGR implementation in DtnSim and ION is loaded with specific features of the algorithm plus compatibility functions which somehow complicates the management of subroutines and hides the inner functioning of the mechanisms. Since the goal of this work is to study the characteristics and scalability of the CGR algorithm and not the compatibility features we created our own lightweight and streamlined CGR for the NanoMind OBC.

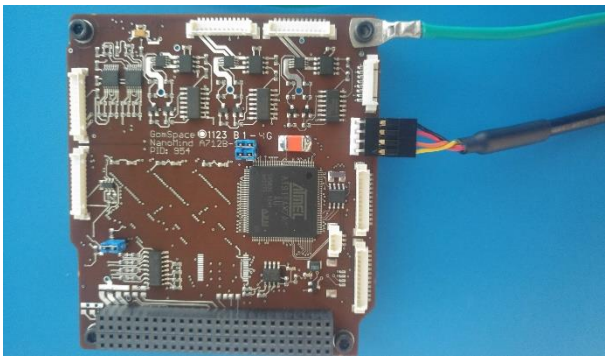

Figure 1: NanoMind OBC on test bench

CGR Implementation for the NanoMind

The CGR algorithm was re-implemented in C language in a condensed and reduced fashion. Although a simplified version of the algorithm compared with ION software, it is capable of fulfilling the tasks explained in [10] (reading a contact plan and meeting the route that responds to the requirements), while able to run on a limited flight computer as the NanoMind A712c.

In a DTN network routed by CGR, the number of nodes does not directly affect the execution time, but the number of contacts between these nodes. Indeed, the algorithm iterates over contacts imprinted in the contact plan, which has to be periodically updated to each of the orbiting spacecraft. However, if the contact is too large, the memory resources of the OBC can be stressed and eventually overloaded.

In particular, the memory limitation is consequence of a linked list implementation of the contacts data structure. Linked lists advantages are crucial for CGR as they are used to dynamically allocate the contact plan's memory space. The number of nodes and contacts can grow (new arrives from contact plan updates) and shrink (old contacts are due and thus removed from memory), so a

contiguous memory storage is not an option for such object.

Even though the linked list approach is well suited for CGR concept of operations, it limits the algorithm capacity of delivering a result swiftly. Indeed, computing a route in CGR may require iterating through most or all of the elements on the list. This makes the algorithm runtime difficult to predict or calculate without an empirical evaluation that leverages low-level software measurement tool-chain provided by the OBC manufacturer. Still, the method used to make the memory measurements is simply a close track of `malloc` calls (memory allocate functions) during the contact storage and route computation phases.

The compilation of code written in C language is done using the ARM toolchain arm-none-eabi-gcc-4.6.4, at 40Mhz as a primary task of the FreeRTOS real-time operating system.

EXPERIMENTAL CAMPAIGN

An exhaustive experimental campaign was conducted on a realistic contact plan comprised of 10 ground stations, 100 ground nodes and 12 satellites as nodes. The contact plan was derived from realistic simulations of a Walker formation of the 12 satellites, specific locations of the ground stations, and random locations for the ground nodes.

For this case study, a Mission Operations and Control (MOC) node that has permanent contact with all the ground stations (node 300) is added to the contact plan. The runs were prepared so that computed routes start at the control node, go through a ground station, from there to a satellite (or more) and then to a ground node as the final destination. It is worth mentioning that the ground nodes will not necessarily run the algorithm on a limited capacity computer in a real application. Anyway, we consider they do in this experiment to further stress the runtime in the Nanomind OBC.

Access computations were executed with AGI's Systems Toolkit software available at CONAE. The output of this data was exported to a contact plan expressed in ION format, the same input structure our NanpMind CGR can read.

The resulting contact plan comprises mor than 9000 contacts between these nodes. Table 1 sumarizes the first 27, including their start and end time, the source and destination node and associated data rate.

The experiment is as follows. First, 10 contacts are loaded into NanoMind's memory, then CGR is executed and the time to deliver the complete route table is measured. Then, the used memory is freed. Next, a total of 20 contacts are loaded, and the process repeats with steps of 10 contacts until the memory is full.

The implemented CGR algorithm delivers all the routes it finds from a source node to a destination node. However, the first route always arrives earlier to the destination in the contact plan. In other words, CGR route computation delivers a series of routes from the contact plan in order, from the best to worst one. Supposedly, the second, third, and so on best routes are valid paths to the destination as the best ones are either consumed as traffic is forwarded or they become old in the sense that the limiting contact (typically the first contact, but not always) in the route is no longer usable (i.e., it has already ended). To avoid routing loops, this method is based on Dijkstra's shortest path algorithm for path selection as proposed by Segui et al. [14]. For more details on the operation of the algorithm the user is referred to [10].

Our software is able to detect memory exhaustion by monitoring the MALLOC_FAILED_HOOK flag from NanoMind software. When the flag is raised, there no more memory, and no more contacts from the contact plan can be allocated. A "Heap is full but trying to allocate 4096!" warning is shown.

Once the memory limit was found, the next step was to change the bundle's destination node to a different one. From the 100 ground nodes, nodes 11 to 70 were used as destination. A sample of the obtained routes are listed in Table 2, for which the software measurements were:

```
Proximate nodes:  1, 9,
Next hop: 9,
Execution time (ms): 108765
Used Mem (bytes): 2678036
Freed Mem (bytes): 2181448
```

The runtime was measured at the following test points:

```
//start counter
portTickType start = xTaskGetTickCount();
//start CGR
cgrForward(bundleP, cp, En);
//stop counter
portTickType stop = xTaskGetTickCount();
printf("Exec time:\t%lu\n",stop-start);
printf("Used memory:\t%d\n", totalMem);
printf("Freed memory:\t%d\n", freedMem);
```

From the above code, it is worth indicating that `xTaskGetTickCount()` is a FreeRTOS function that measures the algorithm execution time. To this end, we have defined `#define config TICK_RATE_HZ (portTickType) 1000)`.

Table 1: Contact Plan Extract

From time	To Time	From Node	To Node	Bit Rate
300	9	0-86400	300	9
6	406	1	121	1
6	406	121	1	1
274	611	90	121	1
274	611	121	90	1
529	835	9	119	1
529	835	119	9	1
641	1014	14	115	1
641	1014	115	14	1
1207	1409	17	116	1
1207	1409	116	17	1
1628	1757	58	115	1
1628	1757	115	58	1
1810	1978	47	119	1
1810	1978	119	47	1
2983	3290	58	118	1
2983	3290	118	58	1
7279	7445	71	118	1
7279	7445	118	71	1
9800	9867	98	115	1
9800	9867	115	98	1
11482	11851	14	112	1
11482	11851	112	14	1
11503	11767	47	111	1
11503	11767	111	47	1
11504	11846	59	111	1
11504	11846	111	59	1

Table 2: Routes Found Extract

From	To	Contact (start time – end time)
300	9	0-86400
9	119	529-835
119	47	1810-1978
47	111	11503-11767
111	59	11504-11846
Next Route		
300	1	0-86400
1	121	6-406
121	90	274-611
90	117	871-1178
117	31	641-973
31	113	1366-1742
113	77	1379-1755
77	116	2084-2409
116	69	2026-2394
69	112	8701-9044
112	59	10056-10432
Next Route		
300	9	0-86400
9	112	0-121
112	19	124-422
19	115	658-982
115	17	2561-2910
17	118	3978-4353
118	69	4892-5268
69	114	5827-6195

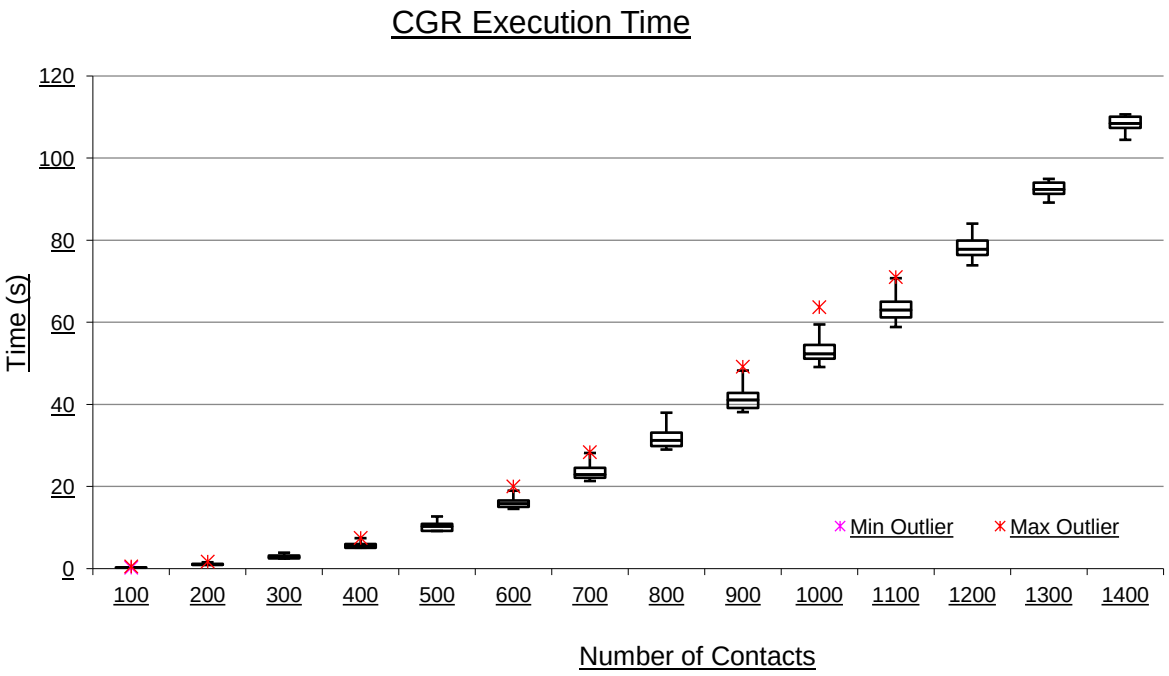

Figure 2: Execution time of CGR running on NanoMind OBC

RESULTS AND ANALYSIS

The main results of the campaign are the statistics of time measurements showed in box plots Figure 2. It can be noticed that, as the contacts between nodes increases, the algorithm execution time increases exponentially. The growth is noticeable from topologies larger than 300 contacts.

Furthermore, CGR was only able to compute routes with contact plans comprised of 1400 contacts. At this contact plan size, the OBC was left with no further memory available to store more data. Also, at this point, the routine required 120 seconds to complete the route table computation. 120 seconds anyways imposes a practical barrier for realistic use cases.

Table 3 shows the ammount of routes found for destination nodes 14 and 19 (those with more routes possible) and 28 and 13 (those with less routes feasible), for each contact plan size. Naturally, the larger the contacts in the topology, the more routes discovered. A color code highlights that a few destination nodes are reachable with up to 13 different paths, while other with only 2. This heterogeneity maps to a diverse execution time in boc plots in Figure 2. It interesting to note that destination nodes 28 and 13 would waste valuable compute cycles for contact plans larger than 500 contacts. Actually, the same happens to node 14 and 19, but this route number is reached with 1000 contacts.

Figure 3 presents the computation time for each of these nodes. This plot shows how route calculation routines that delivers more routes actually consume more processing power.

	Routes found in CGR														
Number of Contacts Used in CGR	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
Dest. Node															
14	1	3	4	5	6	8	9	10	11	13	13	13	13	13	13
19	1	2	4	5	6	8	9	10	11	12	12	12	12	12	12
28	0	1	2	2	2	2	2	2	2	2	2	2	2	2	2
13	0	0	1	1	2	2	2	2	2	2	2	2	2	2	2

Table 3: Number of routes found.

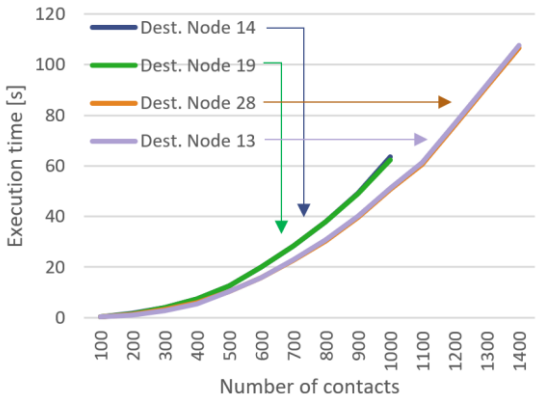

Figure 3: Execution time for nodes 14, 19, 28 and 13.

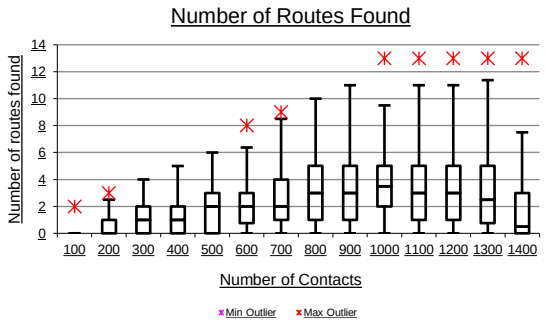

Figure 4: Number of routes found on NanoMind OBC

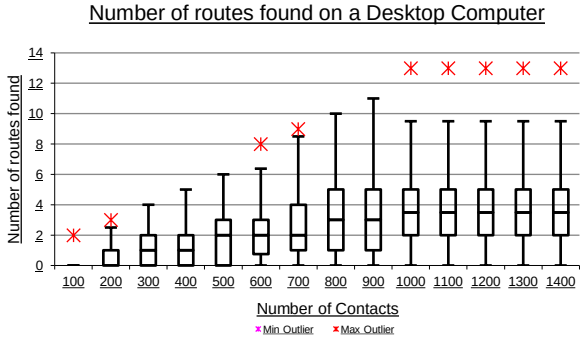

Figure 5: Number of routes found on a Desktop PC

Figure 4 presents the number of routes found for all nodes, not just the extremes presented for 14, 19, 28 and 13. From the figure is clear, however, that the metrics for the latter nodes are actually on the edges of the presented box plots. In general, as the number of contacts increases, more routes become available for the destination nodes, on average.

However, the number of routes founds stagnate starting from 900 contacts onwards. From this point on, we discovered that for some destination nodes the OBD runs out of memory. Thus, only a given set of routes are delivered by the algorithm, although more seem to be possible in the contact plan. Figure 5 shows the same CGR execution on y PC with unbounded memory. The plot evidences some routes after 900 contacts are just missed by the OBC. We discover this is due to the failures on memory allocation attempts. Finally, at 1400 contacts, no more routes are found for any node as the OBC run out of memory even for those nodes with the least number of routes.

Profiling

To gain more insights on the details, a code profile analysis was performed with Gprof [15] software. Results showed that the most frequently called functions are related to linked list creations and element search within them. In the third place comes a Dijkstra's internal process that solves the shortest transmission times. Table 4 shows the main differences in two case studies.

Number of function calls		
Function	Destination 13 (less routes found)	Destination 14 (more routes found)
dijkstra	3	14
createNeighborList	3	14
addToNodeList	257139	254010
isInNodeList	34179	33871

Table 4: Calls to most relevant functions.

Profiling threw some light to the dependency of the execution time against the number of discovered routes indicated in Figure 3. Interestingly, a function called `CreateNeighborList` exhibited a predominant time consumption in nodes for which a large number of routes were found (i.e., node 14 and 19). Internally, this routine is in charge of ordering the contact plan so that it is prepared to be analyzed by the Dijkstra program. The more contacts in the plan, the longer execution time for this function. Plus, the more routes found the greater number of calls to this routine. The reason for the latter is that Dijkstra is restarted from scratch every time a new route is found by CGR. As a result, the overall time consumption of CGR is proven dependent of the size of the computed route table.

We believe there is room for optimizing this aspect. In particular, the contact plan can be prepared only once even if a route-finding process is bootstrapped from scratch. We, however, understand that current CGR implementation in ION and DtnSim is configured in such sub-optimal fashion: in the practical case, only a single route is computed. Once depleted or due, another route is computer for the destination. Since there can be a considerable time-gap in-between both routes computation, the memory occupied by the prepared contact plan should be better released. An optimization at this stage can still be advantageous when multiple routes are needed at the same time (i.e., redundancy or load balancing).

The graph on Figure 6 provide further detail on the memory utilization. In particular, the allocated memory counter is registered every time the `malloc` function is

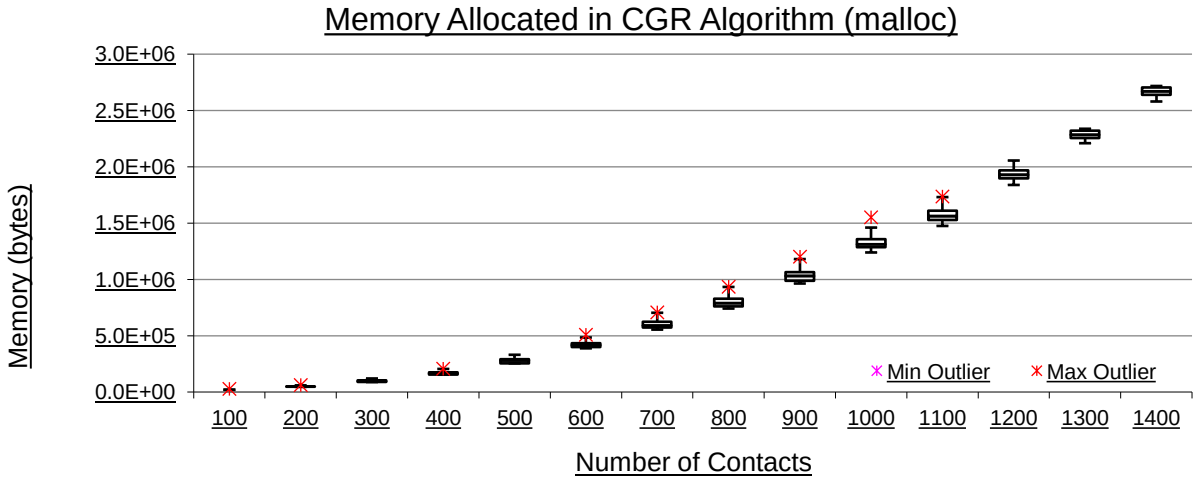

Figure 6: Allocated memory in CGR execution

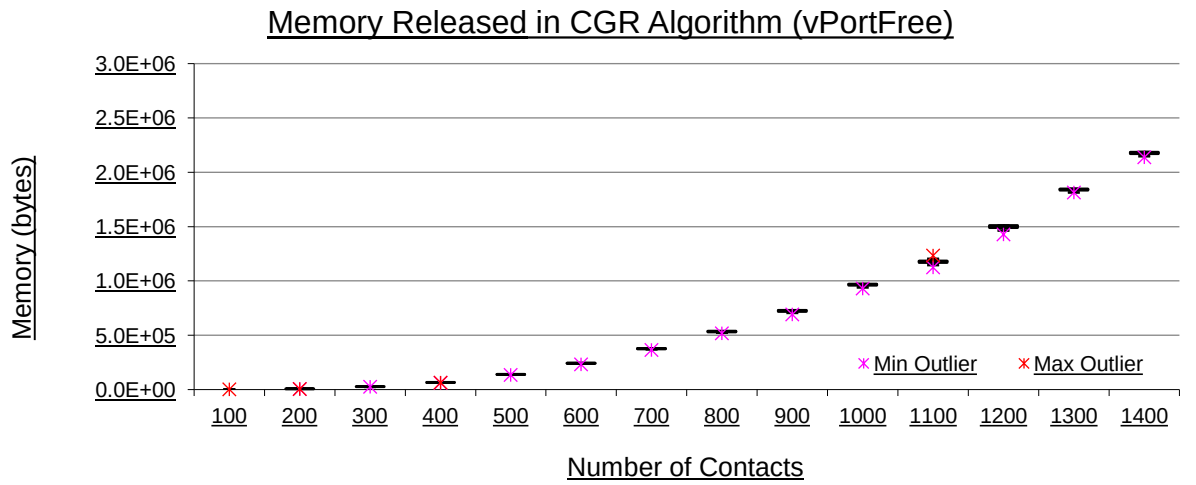

Figure 7: Released memory in CGR execution.

called. Next, some of this memory should be released after a route found and algorithm returns. The released memory graph in Figure 7 shows the sum of the amount of memory that is made available to the operating system inside the `vPortFree` function by a counter called `memfreedcount`, which is called after the algorithm resolves the path.

The `vPortFree` is function responsible for counting each time residual memory is released or deallocated. Fragmentation is not taken into account here, when using the code heap_3.c making `malloc` and `free` thread-safe by temporarily suspending the FreeRTOS scheduler as Memory Allocation Scheme. The comparison of Figures 6 and 7 gives an idea of the data at the output of the algorithm that is deliver to routing table processing stage (assuming no memory leaks are present).

DISCUSION

To put these results in a nanosatellite platform context, consider that traditional Attitude Determination and Control Subsystems (ADCS) algorithm execution times are in the order of 207 µs [16] in the same platform, as investigated by the Argentinian Space Agency. Moreover, this routine has a periodicity of 1 second. Thus, in principle, if there are no other important tasks on the satellite OBC (i.e., thermal control and payload data management), then CGR execution seems to be feasible. In particular, we internally discuss that a proper

scheduling of the route computation power can be prepared in advance so that it can run in the background, with lower priority as the links become available, as already considered by the DTN architecture.

Moreover, there is plenty of room for optimizing the current implementation of CGR used for this experiment. On the memory side, we only used the RAM memory of the OBC, disregarding other external memory sources (i.e., SD memory slots are available). Further research on leveraging such a slow yet larger memory bank can bring some practical alternatives to the highlighted issues. On the processing side, the route is logged and transmitted to the host PC at the end of the building route list process. The print function affects the execution time and the memory stack, but the effect is minimal since the serial transmission is at a speed of 115200 baud. We will look after improved logging techniques for future experiments. Furthermore, the algorithm is executed directly on the flight software as an operating system task. The NanoMind FreeRTOS operating system represents 1% of processor utilization, reported as an IDLE task, which can be reduced to the minimum with specific OS configurations.

CONCLUSION

In this work, an analysis of the Contact Graph Routing (CGR) algorithm execution time and memory utilization was performed on an on-board computer for cubesats. A realistic contact plan and a real computer were leveraged and brought into a testbed for processing and memory benchmarking. To this end, we implemented our own lightweight CGR algorithm in C language to run in FreeRTOS. To the best of author's knowledge, this is the first DTN routing evaluation on a nanosatellite OBC.

The experiment has shown what the hardware is capable of running and delivering routes correctly with contact plans up to 900 contacts. The limited amount of memory is the principal limitation, although processing time at this topology size is already in the order of 40 seconds for a single destination. Future work will be focused on better memory management in FreeRTOS and on code optimizations to improve the computation time. In the long term, we have the expectations of running in-orbit experiments with Argentinian Cubesat missions to validate the approach of constructing cheap and disruption-tolerant LEO DTN satellite constellations.

Acknowledgments

This study was conducted in the classrooms and laboratories of the "Unidad de Formación Superior – UFS, CONAE" during the period 2019-2020 as part of the thesis project of the master's degree in Satellite Instruments thanks to a full-time scholarship. Argentine Government sponsorship is acknowledged.

Special thanks to all MIS members, Javier Uranga and Marco Alvarez Reyna for collaboration and support.

References

[1] S. C. Burleigh, T. De Cola, S. Morosi, S. Jayousi, E. Cianca y C. Fuchs, «From Connectivity to Advanced Internet Services: A Comprehensive Review of Small Satellites Communications and Networks,» *Wireless Communications and Mobile Computing*, vol. 2019, nº 6243505, 02 05 2019.

[2] E. Kulu, «Nanosats Database,» [En línea]. Available: https://www.nanosats.eu/. [Último acceso: 10 06 2020].

[3] T. Villela, C. A. Costa, A. M. Brandão, F. T. Bueno y R. Leonardi, «Towards the Thousandth CubeSat: A Statistical Overview,» *International Journal of Aerospace Engineering*, vol. 2019, nº 5063145, 10 01 2019.

[4] K. I. o. Technology, «Official Website for BIRDS project,» 2017. [En línea]. Available: https://birds-project.com/. [Último acceso: 10 06 2020].

[5] T. M. Lovelly, B. Donavon, K. Cheng, R. Kreynin, A. D. George, A. Gordon-Ross y G. Mounce, «A Framework to Analyze Processor Architectures for,» *IEEE Aerospace Conference*, pp. 1-10, 2014.

[6] R. Some, R. Doyle, L. Bergman, W. Whitaker, W. Powell, M. Jhonson, M. Goforth y M. Lowry, «Human and robotic space mission use cases for high-performance spaceflight computing,» *AIAA Infotech@Aerospace (I@A) Conference*, Vols. %1 de %2AIAA 2013-4729, nº Aug, 19-22 August 2013.

[7] U. o. C. Scientists, «UCS Satellite Database,» 1 April 2020. [En línea]. Available: https://www.ucsusa.org/resources/satellite-database. [Último acceso: 10 06 2020].

[8] J. A. Fraire, S. Céspedes y N. Accettura, «Direct-To-Satellite IoT - A Survey of the State of the Art and Future Research Perspectives,» *Ad-Hoc, Mobile, and Wireless Networks. ADHOC-NOW*

2019. Lecture Notes in Computer Science, vol. 11803, pp. 241-258, 25 09 2019.

[9] D. Gershgorn, «POPULAR SCIENCE,» 17 08 2015. [En línea]. Available: https://www.popsci.com/samsung-wants-launch-thousands-satellites-bring-everyone-earth-internet/. [Último acceso: 10 06 2020].

[10] J. A. Fraire, P. Madoery, S. Burleigh, M. Feldmann, J. Finochietto, A. Charif, N.-E. Zergainoh y R. Velazco, «Assessing Contact Graph Routing Performance and Reliability in Distributed Satellite Constellations,» *Journal of Computer Networks and Communications*, vol. 2017, nº 2830542, 2017.

[11] S. Burleigh, A. J. Hooke, L. Torgerson, K. Fall, V. G. Cerf, B. Durst, K. Scott y H. Weiss, «Delay-tolerant networking: an approach to interplanetary Internet,» *IEEE Communications Magazine*, vol. 41, nº 6, pp. 128-136, June 2003.

[12] «FreeRTOS™ Real-time operating system for microcontrollers,» [En línea]. Available: https://www.freertos.org/. [Último acceso: 8 2019].

[13] J. A. Fraire, P. Madoery, F. Raverta, J. Finochietto y R. Velazco, «DtnSim: Bridging the Gap between Simulation and Implementation of Space-Terrestrial DTNs,» *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pp. 120-123, 2017.

[14] J. Segui, E. Jennings y S. Burleigh, «Enhancing Contact Graph Routing for Delay Tolerant Space Networking,» *IEEE Global Telecommunications Conference- GLOBECOM 2011*, pp. 1-6, 2011.

[15] S. L. Graham, P. B. Kressler y M. K. McKusick, «Gprof: a call graph execution profiler,» *SIGPLAN Not*, vol. 39, nº 4, p. 49–57, April 2004.

[16] J. Garrido, J. Zamorano y J. A. De la Puente, «Static analysis of WCET in a satellite software subsystem,» *OpenAccess Series in Informatics*, vol. 30, pp. 87-96, 01 2013.