Enhanced Pathfinding and Scalability with Shortest-Path Tree Routing For Space Networks

Olivier De Jonckère* Juan A. Fraire†‡, Scott Burleigh§
*Technische Universität Dresden, Dresden, Germany
†Univ Lyon, Inria, INSA Lyon, CITI, F-69621 Villeurbanne, France
‡CONICET - Universidad Nacional de Córdoba, Argentina
§ D3TN U.S. Corp., Florida, 444 Brickell Avenue, Miami, FL 33131, USA

***Abstract*—Contact Graph Routing (CGR) is the state-of-the-art deterministic routing approach for scheduled space Delay-tolerant Networks (DTN). Indeed, CGR outlined the Schedule-Aware Bundle Routing (SABR) recommended standard from the Consultative Committee for Space Data Systems (CCSDS). The core approach exploits the predictability of space node mobility patterns and link availability, which is imprinted in time-varying graphs fed to Dijkstra and Yen's algorithms for route computation (pathfinding). This paper addresses the scalability issues CGR faces when computing routes over large-scale contact plans spanning several nodes and long-term planning horizons. After an in-depth analysis of the main CGR computation limitations, we propose multipath-tracking and tree-caching embedded in the Shortest-Path tree routing for Space Networks (SPSN) routing scheme. Simulation results show that SPSN's route computation time does not increase with increasing contact plan horizon while slightly improving on CGR's delivery rate.**

***Index Terms*—Contact Graph Routing, Shortest-Path Tree Routing for Space Networks, Delay-Tolerant Networks, Schedule-Aware Bundle Routing**

I. Introduction

Derived from Contact Graph Routing (CGR) [1], Schedule-Aware Bundle Routing (SABR) [2] is standardized by the Consultative Committee for Space Data Systems (CCSDS) as the recommended routing approach in space network with predictable topologies. Furthermore, SABR is identified as essential for the service management function by the Interagency Operations Advisory Group (IOAG), as noted in the report for the future Mars Communications Architecture [3].

In the context of SABR, a "route" is a list of successive forwarding episodes ("hops") from a source to a destination node. SABR defines a procedure for route selection among a list of pre-computed routes stored in a routing table. The strategy by which the routing table is populated is therefore critical, not only for accuracy but also from a processing time perspective. In particular, recent research showed CGR computational complexity can increase beyond practical limits in large topologies [4], [5].

The computational load of CGR derives in part from Yen's algorithm complexity but is exacerbated by other considerations that are so far only partially understood. In this paper, we expose and dive deep into these undetected and intricate CGR peculiarities including a) path distance metric, b) optimal path identification requirements, and c) ordered route computation in Yen's algorithm.

The remainder of this paper is organized as follows. Section II reviews the operational context of scheduled DTN and CGR, including a characterization of the issues specific to space DTN pathfinding and route management. Section III introduces multipath-tracking and tree-catching in the context of the Shortest-path Tree Routing for Space Networks (SPSN). Section IV discusses the simulation results that validate SPSN is able to improve CGR's state of the art. Conclusions are summarized in Section V.

II. Background

A. Space Delay-Tolerant Networking

A space DTN is characterized by lengthy end-to-end delays, due to both the intermittent nature of the links and the signal propagation delays resulting from operating communication links over interplanetary distances.

Support for tolerating disruptions and delays is provided by the DTN architecture [6] and more specifically by the bundle protocol (BP) [7]. The entities that issue, forward, and consume the data flowing through a BP network are termed *nodes*. BP relies on nodes' ability to store messages while awaiting onward transmission opportunities. Protocol data units of arbitrarily large and variable size, called *bundles*, flow through an overlay network layer implemented by BP. ("Bundles" are so named because they commonly bundle data together with any metadata that the data recipient might need in order to consume the data productively, eliminating conversational negotiation of transmission parameters that is undesirable over lengthy end-to-end communication delays.) The protocols underlying BP, at what is termed the "convergence layer", may form one or more sub-networks of the DTN network or may constitute lower-layer communication links.

In a scheduled DTN, the intervals of communication opportunity between the nodes, called *contacts*, are known in advance and can be enumerated in documents called "contact plans". A contact is a unidirectional transmission opportunity between two nodes, for a predicted time interval, characterized by a data transmission rate also known *a priori*. Contacts commonly (but not always) occur in pairs, constituting bidirectional communication. The contact plan also notes the distance in light seconds (the "range", equating to signal propagation delay over a direct link) between any two nodes for which contacts are predicted.

B. Contact Graph Routing

CGR presents a route construction process and a route selection process. The route selection process is described by the SABR standard, while only suggestions and terminology are provided for the route computation process.

The route computation process can be further divided into two mechanisms: the shortest-path search procedure, implemented by an adaptation of Dijkstra's algorithm, and the alternative route search procedure, typically implemented by an adaptation of Yen's algorithm (as suggested by the SABR standard and implemented within the Interplanetary Overlay Network (ION) [8] software distribution).

The shortest path from a source node to a destination node, as found by Dijkstra, will minimize the arrival time of a bundle in the most general case: the algorithm explores the graph from the source node, starting at the current time, for an abstract bundle of size zero, and any prior allocation of transmission capacity to other bundles is ignored. But the properties of a given bundle may make it impossible for that bundle to be forwarded on that shortest path: bundles already allocated to that path may leave too little residual transmission opportunity for the bundle, given its size. In this event, the shortest suitable alternative path must be found.

Yen's K-shortest-path algorithm may be used for this purpose, but it suffers from polynomial complexity and invokes Dijkstra's algorithm within its inmost nested loop. This compound complexity, amplified by the polynomial complexity of Dijkstra's algorithm itself, contributes significantly to CGR's computational cost [5].

A first step toward reducing this cost is to reduce the Dijkstra algorithmic complexity by replacing the contact graph with a node graph: instead of a graph in which the vertices are contacts (between nodes) and the arcs are periods of bundle retention between contacts, we have a graph in which the vertices are nodes and the arcs are lists of contacts between nodes. This modification has proven to be effective, as reported in [9] and [10].

A further problem with the use of Yen's algorithm is the difficulty of selecting a value for K, the number of alternative shortest paths to compute: K cannot be infinite, but if K is too small, then none of the computed paths may be suitable for forwarding a given bundle. So the second step toward computational cost reduction in space DTN routing is to compute new routes only when needed. We can "set" K to infinity but "pause" the algorithm until a search for a suitable route for a given bundle is unsuccessful, and then "resume" it only until one suitable route has been inserted into the table. This adaptive Yen mechanism was implemented in ION to reduce computational pressure during initial route computation and subsequent route selection.

Additionally, two alternative CGR mechanisms were developed in [11]. One approach was modifying the Dijkstra search to consider bundle size and contact capacity consumption while searching to enable route computation and selection in a single operation. The other approach was the replacement of Yen's algorithm with a contact-limiting mechanism, suppressing one contact before each iteration of the path-finding loop. This enabled another route to be found in a single invocation of Dijkstra. In contrast, the number of Dijkstra invocations required by Yen's algorithm was approximately equal to the average path length. However, alternative routes that included any of the suppressed contacts were rendered undetectable. Finally, authors in [12] presented hop count as an alternative CGR optimization method.

C. DTN pathfinding challenges and issues

It is commonly assumed that the limit on the scalability of CGR is its sensitivity to contact plan length. However, reducing the sensitivity to the contact plan length as in [10] is not sufficient. Instead, the use of Yen's algorithm to find alternative routes remains a main contributor to computation cost in large intermittently-connected space network networks [9]. In the following, we discuss the main CGR scalability issues.

a) Path distance metric: The first peculiarity is the definition of the path distance in the context of SABR. An analogy would be the booking of train tickets between two train stations. One customer might prefer to minimize the transit time and book train tickets with fewer connections by delaying his departures to catch direct trains. A second customer might prefer to minimize the arrival time, even if this implies more connections. The arrival time of this customer might be constrained by the connections between the final station and the previous one, if the destination is less connected to the train network, i.e. connections for this destination are less frequent.

SABR behaves in the latter manner, by always prioritizing earlier arrival times when selecting a route. The arrival time at a destination might be heavily constrained by the contacts between the penultimate and the last node (the destination) of the path. If such a configuration occurs, an arbitrarily high number of paths sharing the last contact of the best route might exist, and thus those routes share the same arrival time (see contact the case of C_1 in figure 1). This statement still holds true if pathfinding takes into account the size of a given bundle to schedule, i.e. the delays between the first and last bytes transmission times.

In the case of the first customer discussed above, we could also find an arbitrarily high number of trips sharing this minimum transit time, but they would be unlikely to share the same last connection. A similar set of connections can be available the next day but would be translated by different contacts in the context of SABR.

This aspect is independent of the algorithmic complexity.

b) Optimal path requirements: The second peculiarity is the importance of maximum utilization of all transmission opportunities. The SABR standard imposes exacting requirements on the determination of the best route for a bundle. First, the best route is the one that results in the earliest arrival time. If two routes result in the same arrival time, the one comprising fewer "hops" (that is, involving fewer forwarding nodes) is considered the best. If two routes are characterized by the same

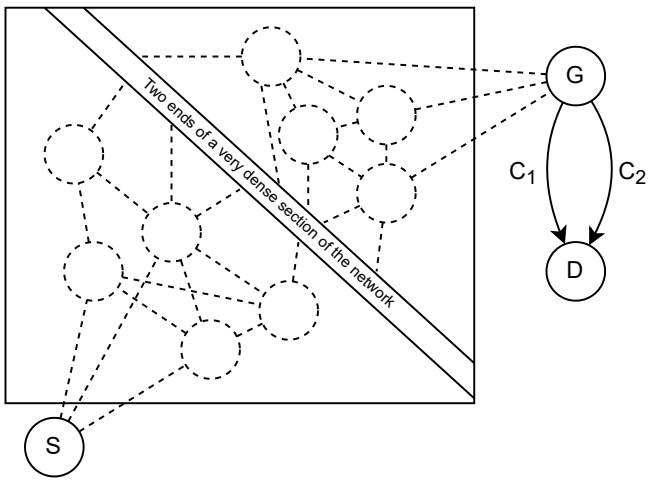

Fig. 1. A very dense network as seen by the local node S, with node D being accessible only via the consecutive contacts C_1 and C_2 from a "gateway" node G. Contact C_2 ends later than C_1, and C_1 ends later than the other contacts.

earliest arrival time and minimal number of hops, the route that expires last (i.e., whose earliest-expiring constituent contact has the latest "stop" time) is best. These rules help maximize the volume of data that can be conveyed by the network while still minimizing the arrival time. But the optimal path cannot always be detected in one search. Instead, a number of searches might be required to find the best route for a given bundle, which may result in excessive computational costs. In Dijkstra's algorithm, the limiting factor when using a node graph is that a vertex can only have a single parent, while in the context of a contact graph, the limiting factor is the interruption of the exploration as soon as the destination is reached (to lower the computational pressure). In both cases, this can inhibit the detection of the best existing path in the sense of SABR.

c) Ordered route computation: The ordered route computation at the core of Yen's algorithm aggravates issues *a)* and *b)* from a compute effort perspective. If the end destination node is reachable via multiple paths that share the last contact (issue *a)*, figure 1), multiple (ordered) Yen's iterations will be needed to revise all paths. However, most of these computations will be performed in vain because higher hop counts and/or limited effective volume limits will violate the optimal path condition in the route selection phase (issue *b)*). If the adaptive strategy implemented in ION is leveraged, this computation loop might block the forwarding of an incoming bundle for several seconds. In this case, the limitation of the "K" Yen's parameter might result in an early termination of the algorithm thus missing the real optimal path.

III. Shortest-Path Tree Routing

In this section, we propose two approaches to overcome the aforementioned issues. The solution is integrated into Shortest-Path tree routing for Space Networks (SPSN). SPSN was introduced in [10] as an alternative route construction mechanism leveraging node multigraphs, volume-aware pathfinding, and shortest-path tree construction. Please refer to this previous publication for details. For shortest-path tree implementations please refer to [13].

A. Multipath-tracking

To address the issue described in section II-C-b), we extend SPSN with a feature coined "multipath-tracking". This feature intends to find routes that might be missed with a single Dijkstra invocation.

The core functionality of multipath tracking is to simultaneously track several routes generated by SPSN's shortest-path tree construction process. For example, in figure 2, the orange and blue paths (respectively the best existing routes to $D1$ and $D2$) are simultaneously tracked in the SPSN tree during a single Dijkstra call. Specifically, node E would carry 2 distinct predecessors for 2 different routes, while a classic Dijkstra exploration in CGR only allows for a single predecessor. This principle is integrated into SPSN as follows.

During the tree construction, the exploration information to a receiver node (parent, arrival time, hop count, expiration, etc.) is not attached to the node graph vertices directly but stored as a "*RouteStage*" work area inserted in a list of "*RouteStages*" for this node. The parent of a *RouteStage* is also a route *RouteStage*. By identifying the best *RouteStage* to a node, the shortest path can be reconstructed from the shortest-reverse-path starting from this *RouteStage* to the *RouteStage* of the source thanks to the parenting information. A shortest-reverse-path tree consequently provides the best *RouteStage* for each reachable destination.

The lists of *RouteStages* enable SPSN to implement multipath-tracking: instead of just replacing the *RouteStage* of a node during exploration. The lists are ordered following the same rules defined for the selection phase of SABR. As mentioned above, a *RouteStage* holds the metrics required to enable such ordering.

Insertion of a candidate *RouteStage* in a list should be allowed in cases where existing *RouteStages* (previously inserted during exploration) in the list are better than the candidate for a higher-ranking selection metric but not for a lower-ranking one.

This mechanism permits to track simultaneously the best paths (first element of each list) without discarding other paths being dependencies of those best paths (e.g. in figure 2, the *RouteStage* to E having the *RouteStage* to S as parent).

To wrap up, multipath-tracking is implemented as a single function that tries to insert a *RouteStage* candidate into the *RouteStage* list of a node. This function also takes care to update the accumulator of Dijkstra's main loop and discard other candidates from the list (and the accumulator) if those *RouteStages* are irrelevant anymore. A diagram depicting a simplified version of this workflow is presented in figure 3.

B. Tree Caching

This section proposes a solution to embed shortest-path tree pathfinding within a routing algorithm. Tree caching is a

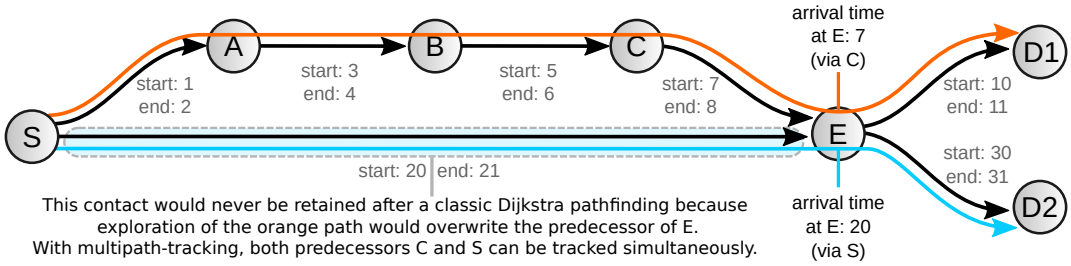

Fig. 2. The orange route is the best route to E and $D1$, showing an earlier arrival time, while the blue route is the best route to node $D2$, showing a fewer hop count. Dijkstra's algorithm constraints parenting to a single vertex, the parent node of E being C. With standard Dijkstra, the blue route is not detected.

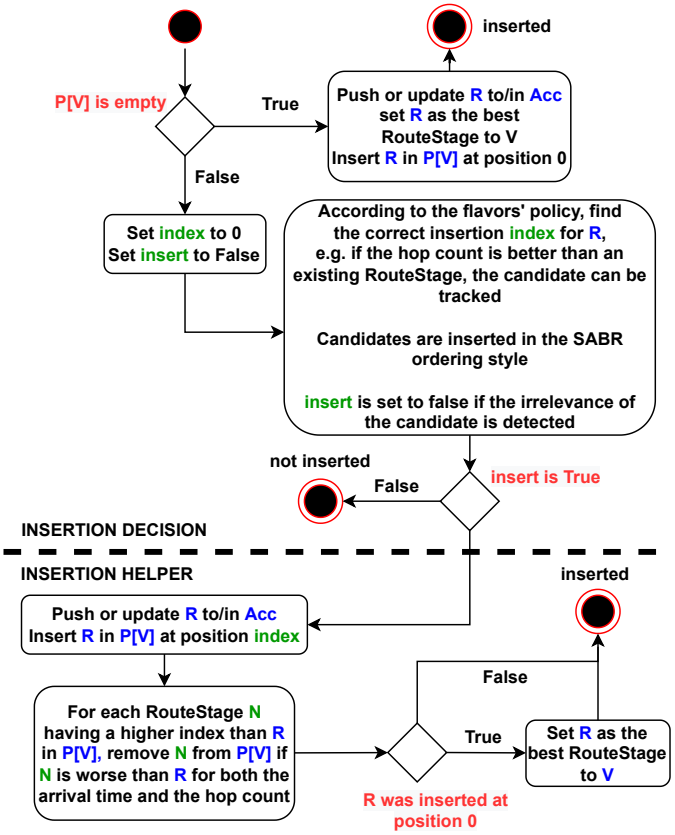

Fig. 3. Insertion mechanism of multipath-tracking. Conditionals are in red, parameters and containers are in blue, and internal variables are in green. $P[V]$ is the list of *RouteStages* to node V. R is a new candidate *RouteStage* to V. Acc the is Dijkstra's priority queue.

Algorithm 1: Adapted version of the $route$ method of the $SpanningTreeRouter$ class from SPSN's python implementation.

Data: $curr\_time$, bundle $destination$, bundle $size$, bundle $expiration$ time, the excluded nodes $exclusions$.
Result: Updates the intervals of available bandwidth and split them if needed along the scheduled bundle path.

if $destination$ **in** $self.size\_max$ **then**
  if $self.size\_max[destination] <= size$ **then**
    return None
if not $size < self.previous\_size$ **and** $self.current\_exclusions == exclusions$ **then**
  if $self.\_schedule(curr\_time, destination, size, expiration)$ **then**
    return $self.\_first\_hop\_contact(destination)$
$self.routes = self.spsn.compute(self.source\_node, curr\_time, size, exclusions)$
$self.current\_exclusions = exclusions$
$self.previous\_size = size$
if $destination$ **not in** $self.routes$ **then**
  $self.size\_max[destination] = size$
  return None
if $self.\_schedule(curr\_time, destination, size, expiration)$ **then**
  return $self.\_first\_hop\_contact(destination)$
return None

replacement for CGR's routing tables in the context of SPSN's trees. The goal is to allow tree reuse to decrease the computational pressure. SPSN implements volume-aware search, similar to the one-route approach from [11], by computing a single tree for a given bundle size. This ensures that no useless Dijkstra invocation will ever be processed. Indeed, a newly computed tree is at least used once for the incoming bundle. On the other hand, as discussed, Yen's solution in CGR might run into multiple useless iterations, complicating the consideration of route volume occupation.

To compute a tree, SPSN requires a bundle size for the volume-aware search and a node exclusion list relative to the bundle destination [10]. The exclusion list serves as a countermeasure against network loops, while the bundle is used to ensure the constructed tree has the required data volume capacity to reach the destination.

The baseline SPSN algorithm computes one tree for each new bundle [10]. However, the core idea behind tree-caching is to allow the re-utilization of previously computed trees for subsequent bundles with different sizes and exclusion list requirements.

To this end, we define a tree as *reusable* for a given bundle if the node exclusion list supplied to SPSN for this tree construction is identical to the exclusion list for this bundle's

destination. Moreover, the required bundle size supplied for the new tree construction shall be less than the bundle size for the existing tree. Indeed, if the existing tree can accommodate a small bundle, an attempt can be conducted for a larger one. However, if the tree was constructed for a large bundle, existing paths suitable for smaller bundles might have been ignored.

However, the tree-caching approach will not choose the best existing path in all cases. As long as a tree is suitable, the tree will be reused, even if an alternative route becomes preferable (booking bundle to a path increase the arrival time for the next bundles booked for the same path). As a result, and similarly to CGR with Yen's, scheduling bundles for a path pushes forward the projected arrival time for the next bundle scheduled for the same path. However, SPSN differs because trees are relevant for all destinations. Specifically, when the tree becomes invalid for a given destination and is replaced, the routes for all destinations shall be updated in forwarding time. In CGR, this can only occur if a route to A exhausts the residual volume of a contact shared by another route to B, triggering the resume of Yen's algorithm for the routing table of B.

Furthermore, the caching mechanism loads and stores trees on an exclusion list basis. If a tree is loaded but unsuitable, it is recomputed, and the new tree replaces the former one for this exclusion list.

A simplified scheduling approach is presented in algorithm 1. It assumes the cache has a single tree (it does not include tree selection based on the exclusion list). The router first checks if failure to find a route has already occurred for a bundle of equal or lesser size. The method would return nothing if this were already the case (lines 1-3). The cached tree can then be used right away if the following criteria are met (line 4):

- The bundle size is superior to the previously required size for tree construction, i.e. verify that no routes are discarded by volume.
- The exclusion list is identical to the exclusion list used when computing the cached tree, i.e. verify that no routes are discarded due to exclusions.

If a forwarding decision is reached, the first hop contact for the current route to the destination is returned (lines 5-6). Otherwise, the cached tree is invalidated, and a new one is computed, with the current time, the current bundle size, and the exclusion list as parameters (line 7). Note that the bundle expiration time is not required, as the fastest route is preferred by design with multipath-tracking. When a new tree is cached, the previously required size and the exclusion list used to create the tree are updated accordingly (lines 8-9). Once the best possible tree is available for the current bundle, a first check is performed to verify that the destination is reachable (line 10). If the node is unreachable, the router will not try to schedule other bundles of equal or greater size for this destination, and the method terminates without choosing a next hop node (lines 11, 12). Otherwise, a new scheduling attempt occurs. In this case, the first hop contact for the current route to the destination is returned, and the algorithm terminates (line 14). Suppose the expiration time is earlier than the computed best arrival time. In that case, the bundle will be found ineligible for forwarding, and the tree-caching method will terminate without choosing a next-hop node (line 15).

IV. Evaluation

A. Simulator

In order to evaluate the multipath-tracking and tree-caching mechanisms, simulations were conducted with *aiodtnsim* [9], [14], [15]. This python simulator permits reproducible simulations with *asyncio*, by providing bundle injection plans at its initialization. Its intuitive design permits the implementation of new routers by inheritance of the available minimal node implementations provided. The platform used for running the simulation is a virtual machine with a 24-core Intel(R) Xeon(R) Gold 6136 CPU @ 3.00GHz.

B. Scenarios

A Ring Road Network (RRN) comprises orbiting satellites and ground assets. RRNs allow delay-tolerant communications with nodes that would be difficult or impossible to serve with wired connections. Indeed, the future Mars networking architecture includes a Martian RRN. The RRN scenarios are generated with the open-source *tvgutil* [16]. This toolkit permits the creation of a) realistic RRN scenarios using CelesTrak CubeSat data [17] and b) bundle injection plans for aiodtnsim. Furthermore, to further stress the size of the network topology, we included urban transport scenarios with a large number of nodes. The list with the specifics of the scenarios is as follows:

1) rr9/10: RRN scenario with 19 nodes, 9 LEO satellites, and 10 ground stations. Two flavors are leveraged (*lw* and *hg*) to study low and high traffic loads.

2) rr15/15: RRN scenario comprised by 30 nodes, 15 LEO satellites, and 15 ground stations.

3) fg: Public transportation scenario from the city of Freiburg composed of 206 nodes, 89 stations, and 117 buses. Three flavors (*5mb+lw*, *1mb+av*, and *1mb+hg*) are used to simulate low, average, and high traffic loads. Moreover, the contact data rates are also 5 times lower for the average and high load sub-scenarios.

Table I summarizes the parameters used for each scenario. Each scenario is simulated with 20 randomly generated injection plans.

C. Algorithms

We compare our solution with the following CGR algorithm flavors, each leveraging a node-based graph structure.

a) cgr-yen-ion: CGR algorithm leveraging an adaptive Yen's algorithm strategy. We mimic its implementation in ION but with configured computational breaks to avoid processing exhaustion. The bundle is dropped after 2 attempts to find an alternative route. The internal Yen's value of K is set with an upper limit of 1000.

TABLE I
SIMULATION PARAMETERS

	rr9/10+lw	rr9/10+hg	rr15/15	fg5mb+lw	fg1mb+av	fg1mb+hg
message sizes (bits)	1638400	16777216	100000	8500000	8500000	45000000
injection interval (secs)	273	112	2	5	3	1
total messages	2500	6150	25000	5750	9500	28700
node count	19	19	30	206	206	206
contact count	10348	10348	3460	16138	16138	16138

TABLE II
SHORTEST-PATH TREE CONSTRUCTION TIME

MPT tree construction mean time in seconds			
Contacts	120000	160000	200000
20 nodes	0.00998	0.00854	0.00826
40 nodes	0.02677	0.02861	0.02908
60 nodes	0.05020	0.04989	0.05592

b) cgr-1st-end: First ending CGR algorithm implementing the limiting contact approach, suppressing the earliest ending contact of the last found route.

c) cgr-cs-1st-end: CGR with contact segmentation and the first ending contact removal is also considered in the analysis. This scheme isolates the impact of contact segmentation, the volume management technique used by SPSN.

d) cgr-cs-hop: CGR-hop, as defined in [12] with contact segmentation. CGR-hop relaxes delivery time to reduce contact utilization (shorter paths in terms of hop count are honored).

We then consider two SPSN flavors based on [18].

e) spsn-mpt: SPSN algorithm with multipath-tracking.

f) spsn-hop: SPSN version of CGR-hop following the same optimization principles than [12], but leveraging SPSN's shortest-path tree construction.

D. Results

Previous publications showed that SPSN was relatively insensitive to the contact plan length [10], suggesting appealing scaling properties. Table II summarizes the primary outcome of our simulations. It proves that this scalability advantage for SPSN still holds when leveraging multipath-tracking regardless of the contact plan horizon.

For the following, we consider simulation time as an indication of the computing effort required for each algorithm. Nevertheless, this time also includes simulation overhead, which is similar in each simulation execution.

Simulation results in figure 4 show that *cgr-yen-ion* uses a mean simulation runtime of 84 hours in the *fg1mb+hg* scenario. Considering ≈25K transmissions took place, an average scheduling decision time of ≈12 seconds per transmission was derived. On the other hand, the mean simulation runtime for *spsn-hop* was less than 7 minutes, leveraging a far larger amount of transmissions (≈75K). This is evidence of the relative performance between the extended SPSN and CGR.

We also observe that all CGR flavors, especially *cgr-yen-ion*, present higher bundle drop rates, as plotted in figure 5, especially for the most challenging *fg1mb+hg* scenario.

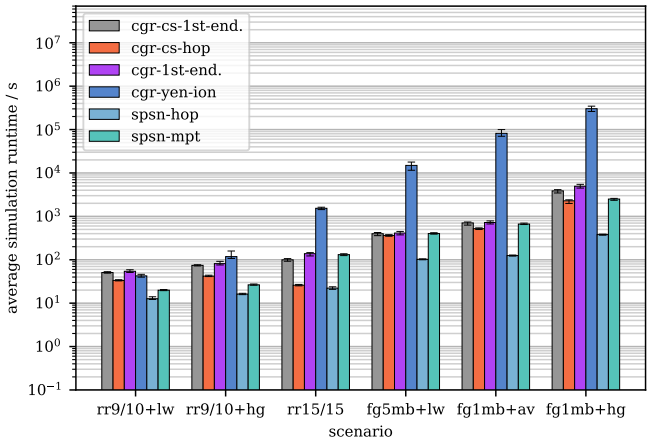

Fig. 4. Simulation runtime (log scale).

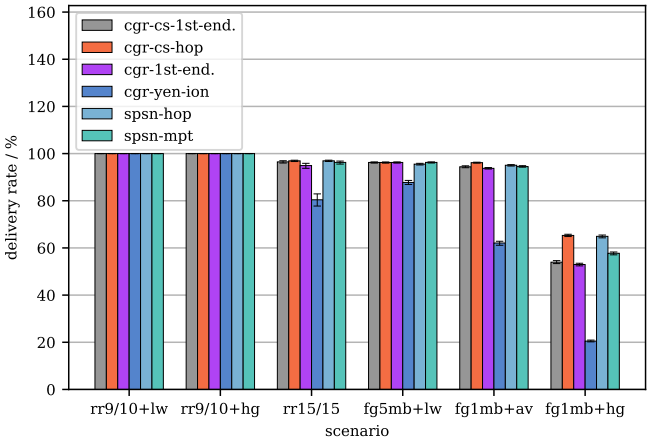

Fig. 5. Delivery rate

Simulation debugging showed that the main drop reason was the incapacity of finding suitable alternative routes. In fact, dropping messages directly at the source reduces the load on the network, giving more space for other messages and thus reducing congestion.

The simulations run-times in figure 4 show that the SPSN flavors are faster when compared to CGR in all scenarios, except *rr15/15*. In the *rr15/15* case, non-Yen CGR approaches are comparable with *spsn-mpt*. We explain this effect by the aforementioned premature bundle drops in combination with the fact that we scale the load with bundle count and not bundle sizes. The premature drops reduce congestion for other bundles, while the relatively early contact plan horizon (24 hours) permits CGR to still show good processing times. In the end, the average runtime per transmission is inferior to 1

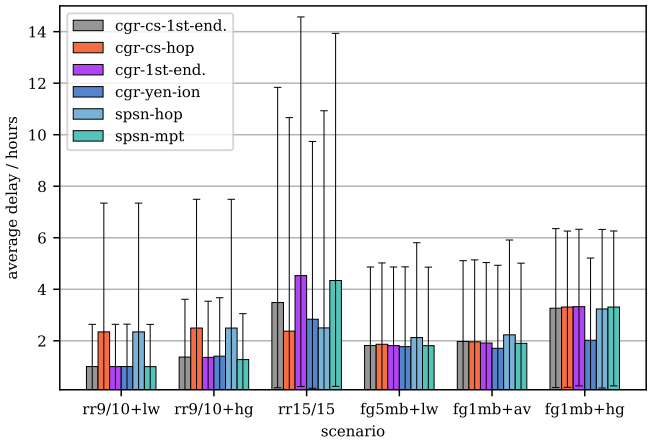

Fig. 6. End-to-end delays.

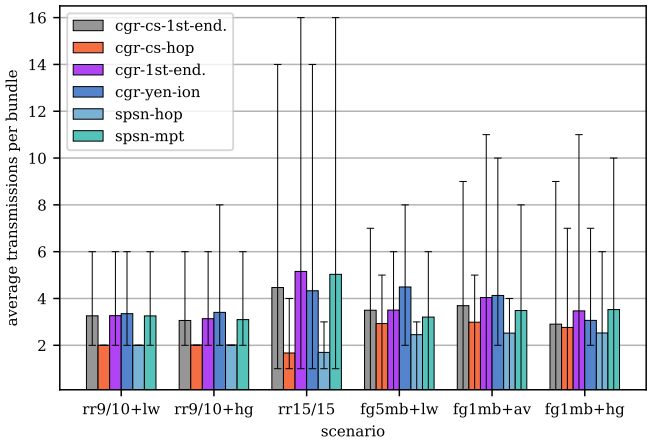

Fig. 7. Average path length.

millisecond for both approaches for this horizon.

On the *fg1mb+hg* scenario, we can observe a clear superiority of the SPSN flavors, with delivery rates of 57.73% against 53.96% for *cgr-cs-1st-end.*, as illustrated in figure 5.

Finally, the end-to-end delays and average path lengths are summarized in figures 6 and 7. We observe that, in general, these metrics are improved with *spsn-hop* and *cgr-cs-hop*. However, the delays can be slightly higher with *spsn-mpt* compared to the non-Yen CGR flavors. The reason for this is that SPSN tries as long as possible to transmit the bundle to the destination, inducing some possible redirections. On the other hand, CGR presents premature drops even though suitable routes exist (but undetectable without Yen's algorithm), reducing virtually the congestion with a reduced contact utilization: 76.15% for *cgr-cs-1st-end* against 85.89% for *spsn-mpt*, both in the *rr15/15* scenario. The *cgr-cs-hop* shows similar delivery rates if compared with *spsn-hop*. Those two approaches do not implement the SABR standard and trade end-to-end delays for reduced congestion. Additionally, the alternative route search for *cgr-cs-hop* is implemented with the first ending approach and therefore suffers from the same premature drops issue.

V. Conclusion

This paper dove into a deep characterization of CGR's scalability issues rooted in the way in which Dijkstra's and Yen's algorithms are utilized. To address the identified issues, we presented multipath-tracking and tree-catching strategies, which have been integrated into SPSN. Simulation results showed that the extended SPSN solution exhibits a more efficient use of the computational effort when compared to legacy CGR, without penalizing other network metrics. The long-term impact of this work involves the consideration of large-scale space DTN topologies, spanning more nodes and longer time horizons. Future work includes the development of alternative SPSN caching strategies to conserve precious transmission opportunities in resource-constrained space networks.

VI. Acknowledgment

This research was conducted with continuous support and counsel from Marius Feldmann and Felix Walter. This work has received support from the Project STARS STICAMSUD 21-STIC-12 Code STIC2020003 and the MISSION project from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 101008233.

References

[1] S. Burleigh, "Contact graph routing," 2009. [Online]. Available: http://tools.ietf.org/html/draft-burleigh-dtnrg-cgr-00
[2] B. Book, "Schedule-aware bundle routing," *Consultative Committee for Space Data Systems*, 2019.
[3] "The future mars communications architecture," *Interagency Operations Advisory Group*, 2022. [Online]. Available: https://www.ioag.org/Public\%20Documents/MBC\%20architecture\%20report\%20final\%20version\%20PDF.pdf
[4] G. Wang, S. C. Burleigh, R. Wang, L. Shi, and Y. Qian, "Scoping contact graph-routing scalability: investigating the system's usability in space-vehicle communication networks," *IEEE Vehicular Technology Magazine*, vol. 11, no. 4, pp. 46–52, 2016.
[5] J. A. Fraire, O. De Jonckère, and S. C. Burleigh, "Routing in the space internet: A contact graph routing tutorial," *Journal of Network and Computer Applications*, vol. 174, p. 102884, 2021.
[6] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Rfc 4838," *Delay-Tolerant Networking Architecture, IRTF DTN Research Group, April*, 2007.
[7] S. Burleigh, K. Fall, and E. J. Birrane, "Bundle Protocol Version 7," RFC 9171, Jan. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9171
[8] S. Burleigh, "Interplanetary overlay network: An implementation of the dtn bundle protocol," 2007.
[9] F. Walter, "Prediction-enhanced Routing in Disruption-tolerant Satellite Networks," Doctoral Dissertation, Technische Universität Dresden, 2020. [Online]. Available: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-721622
[10] O. De Jonckère and J. A. Fraire, "A shortest-path tree approach for routing in space networks," *China Communications*, vol. 17, no. 7, pp. 52–66, 2020.
[11] J. A. Fraire, P. G. Madoery, A. Charif, and J. M. Finochietto, "On route table computation strategies in delay-tolerant satellite networks," *Ad Hoc Networks*, vol. 80, pp. 31–40, 2018.
[12] F. D. Raverta, J. A. Fraire, P. G. Madoery, R. A. Demasi, J. M. Finochietto, and P. R. D'argenio, "Routing in delay-tolerant networks under uncertain contact plans," *Ad Hoc Networks*, vol. 123, p. 102663, 2021.
[13] B. Y. Wu and K.-M. Chao, *Spanning trees and optimization problems*. Chapman and Hall/CRC, 2004.
[14] F. Walter and M. Feldmann, "Leveraging Probabilistic Contacts in Contact Graph Routing," in *IEEE Global Communications Conference (GLOBECOM)*, Waikoloa, HI, USA, 2019.
[15] https://gitlab.com/d3tn/aiodtnsim.
[16] https://gitlab.com/d3tn/dtn-tvg-util.
[17] http://www.celestrak.com.
[18] https://bitbucket.org/olivier-dj/pyspsn.