Scalable Schedule-Aware Bundle Routing

Olivier de Jonckère

TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Chair of Computer Networks

Scalable Schedule-Aware Bundle Routing

Dipl.-Inf Olivier De Jonckère

Born on: 24th May 1991 in Boulogne-Billancourt

Dissertation

to achieve the academic degree

Doktor-Ingenieur (Dr.-Ing.)

First referee
Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Second referee
Prof. Carlo Caini

Advisor
Prof. Dr. Christoph Sommer

Supervisor
Dr.-Ing. Marius Feldmann

Submitted on: 27th April 2023
Defended on: 22nd June 2023

Abstract

This thesis introduces approaches providing scalable delay-/disruption-tolerant routing capabilities in scheduled space topologies. The solution is developed for the requirements derived from use cases built according to predictions for future space topology, like the future Mars communications architecture report from the interagency operations advisory group. A novel routing algorithm is depicted to provide optimized networking performance that discards the scalability issues inherent to state-of-the-art approaches. This thesis also proposes a new recommendation to render volume management concerns generic and easily exchangeable, including a new simple management technique increasing volume awareness accuracy while being adaptable to more particular use cases. Additionally, this thesis introduces a more robust and scalable approach for internetworking between subnetworks to increase the throughput, reduce delays, and ease configuration thanks to its high flexibility.

Acknowledgments

I would like to express my gratitude to everyone who supported me during my doctoral studies. Firstly, I thank Dr. Marius Feldmann for supervising me all those years and for his mentoring in research and beyond. He introduced me to the topic of Delay-Tolerant Networking with meaningful projects, which allowed my interest in the field to increase in a neverending fashion. I would like to thank him for stating that my research deserved to be continued with a doctorate. This thesis would have never been conducted otherwise. Last, I would like to thank him for his trust, allowing me to start my doctoral studies for a limited time frame.

In the same way, I would like to gratefully thank Prof. Alexander Schill for granting me the opportunity to conduct my research in this time frame, for his support, and for his detailed feedback on my work. I would like to thank him for his thorough review of the thesis manuscript.

I am also highly honored that Prof. Carlo Caini accepted to be the second referee of this thesis, as his work and expertise in space DTN routing deeply inspired the direction I wanted to take for my research. I sincerely thank him for his detailed review and for inviting me to meet his team and present my work at the University of Bologna.

I sincerely thank my former teachers for developing my interest in science, and especially Annabelle Demule Thenon, for pushing me to take a path I would not have otherwise. I would also like to thank Juan Fraire, Felix Walter, and Scott Burleigh from D3TN for the fruitful discussions on DTN and other research topics.

I sincerely thank my partner Orianne Bargain for encouraging me to start this work and for her continuous support until completion.

Finally, I am also grateful to my family and friends for supporting me throughout all those years. I especially thank my parents for their presence despite the distance and my friends for supporting me in any situation since I arrived in Dresden.

Olivier De Jonckère
Dresden, July 2023

Statement of authorship

I hereby certify that I have authored this thesis entitled *Scalable Schedule-Aware Bundle Routing* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, 27th April 2023

Dipl.-Inf Olivier De Jonckère

Contents

List of Figures

List of Listings

1 Introduction

1.1 Motivation

In space networks, end-to-end communications are challenging due to frequent disruptions and high delays along the end-to-end path. In addition to these delays due to disruptions, interplanetary distances induce delays on the links, resulting from the signal propagation speed limit.

The so-called Delay-/Disruption-Tolerant Networking (DTN[1]) architecture tackles those constraints. The DTN messages, called bundles, can be stored at intermediary nodes until the next transmission opportunity occurs, contrary to traditional ground networks, where nodes directly forward the bundles upon reception.

Besides disruptions due to failures or other unplanned events, space networks show challenging topological dynamics due to the mobility of the nodes. Consequently, a network node can communicate with different subsets of neighbors over time. This high constraint makes most of the traditional routing algorithms inapplicable.

Space network connections are often scheduled, i.e., the episodes of connectivity between nodes, called contacts, are known in advance. Thanks to node trajectory prediction and mission planning, operators can create a contact plan, which lists the future contacts that should be considered for end-to-end pathfinding. Several neighbors might be reachable simultaneously, but if only one connection at a time is possible due to hardware constraints (e.g., the satellite operates with a single steerable antenna), mission planning is essential to define potential connections that should be ignored. Routing algorithms can then derive a time-varying graph from the contact plan and find a route to a bundle destination with a delay-tolerant version of shortest path algorithms.

Contact Graph Routing (CGR) [Bur09], later standardized by the Consultative Committee for Space Data Systems (CCSDS) as Schedule-Aware Bundle Routing (SABR) [CCS19] is the reference deterministic routing algorithm in scheduled space networks. The Interplanetary Overlay Network (ION) [Bur07] is a reference NASA implementation to allow nodes to operate within a DTN architecture [CBH+07] and uses CGR as the sole routing algorithm. ION is flight proven and operationally used, e.g., for communications with the international space station. Moreover, CGR is still the exclusive routing algorithm available in ION, attesting to its huge

This thesis is also available at `https://bitbucket.org/olivier-dj/diss/`

[1] For convenience, this acronym is also used in this thesis to refer to networks with delay-tolerant capabilities.

predominance for space DTN.

The place CGR holds in the DTN routing field encouraged, for more than a decade, various research works around CGR to provide enhancements. Those advancements still need to be improved regarding the space network growth predictions.

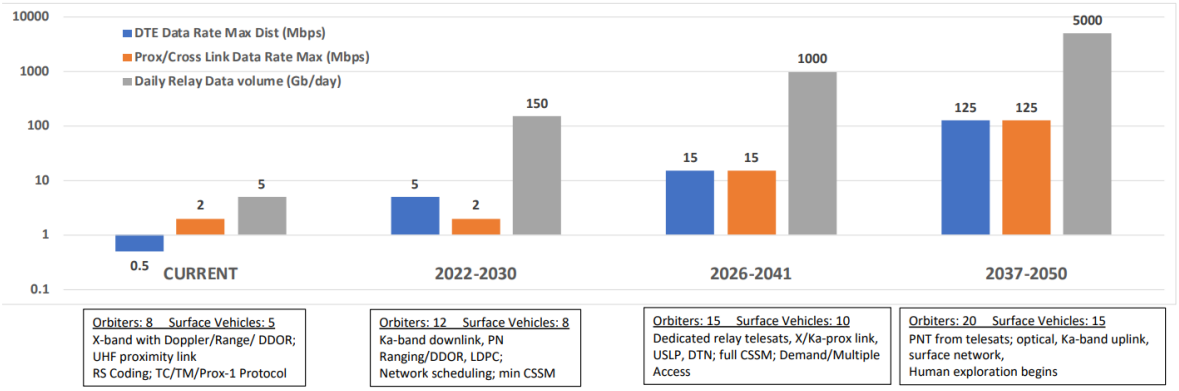


Figure 1.1: Projections of node counts and networking capabilities from the future mars communications architecture report (figure from [ioa22]), page 86.

The future Mars Communications Architecture [ioa22] report from the Interagency Operations Advisory Group (IOAG) published in February 2022 states that each relay orbiter, the user vehicles (in orbit or on the surface), the relevant Earth stations, and the various Mission Operations Centers (MOCs) will serve as DTN nodes. The IOAG also identifies SABR as essential for the service management function.

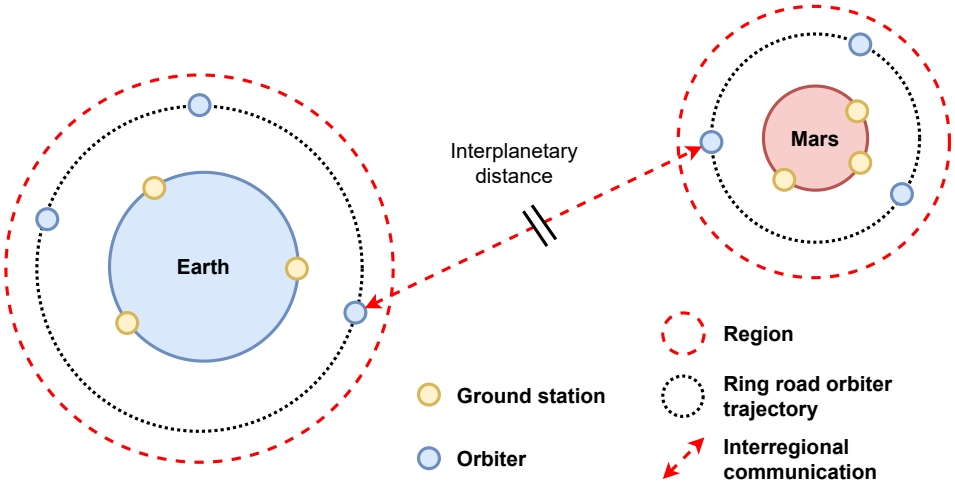


Figure 1.2: Topology-based regional structure, with one region per planet.

However, such a statement conflicts with CGR's known limitations, e.g., its processing time in large networks or its memory pressure when using Yen's algorithm [Yen71, ABB+15, FDJB21, FMCF18, Wal20, WBW+16], rendering unrealistic the ability of CGR to support the networking characteristics presented in figure 1.1.

In response to the predictions regarding the network size increases, sub-networks were also introduced under the name of regions [Ale19], where bundles can flow from one region

to another only through gateway nodes called passageways. Regions permit the reduction of the scope of CGR's applicability to a subset of nodes to mitigate its scalability issues. However, this brings the risk of transferring this issue from the intraregional level to the interregional one, with the multiplicity of small regions addressing the inability of CGR to operate in larger ones. Figure 1.2 depicts a regional structure.

Last, the maximum amount of transmittable data via a contact, called volume, is a precious resource that needs to be managed as efficiently as possible. The contacts can be infrequent and short. The predicted load and bundle size increase (as permitted by DTN technologies) might stress CGR even further when considering existing gaps in its volume management.

The primary motivation for this thesis is the need for a scalable solution to address increasing loads, increasing network sizes in terms of the number of nodes and contacts, and upcoming interregional routing challenges for scheduled space networks.

1.2 Problem statement

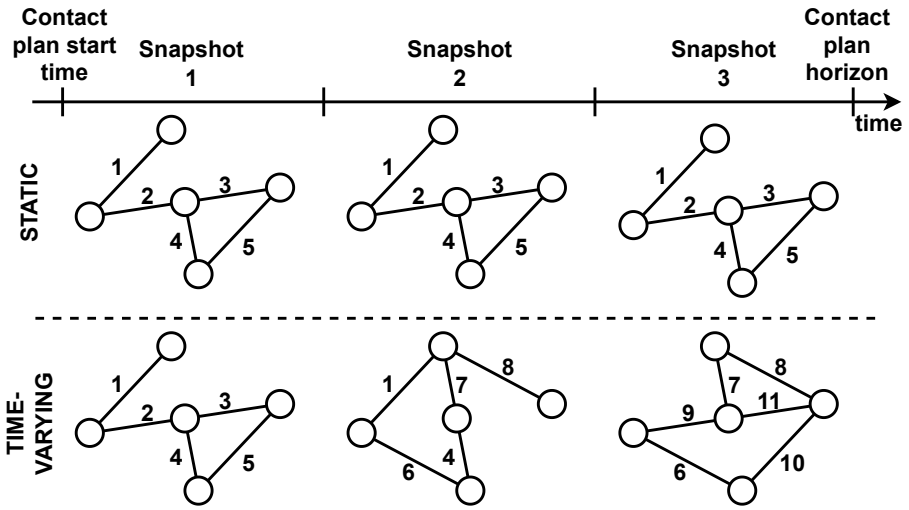


Figure 1.3: DTN contact plan growth compared with a static network.

The space network growth prediction regarding nodes, contacts, load, and delays due to interplanetary links and deep space missions requires efficient routing, volume management, and internetworking solutions.

Routing is a central component to reach this goal, as selecting efficient end-to-end paths can mitigate end-to-end delays by approaching optimal pathing. The multiplication of edges in such network graph representations brings scalability challenges. A static network sees its graph representation increase when the number of nodes and possible contacts increases. At the same time, a DTN graph grows even larger depending on the operational period supported by the contact plan for a given number of nodes. For a given operating period, the horizon of a contact plan refers to the latest contact end time. Figure 1.3 depicts this aspect by considering three different snapshots for two networks and providing the contact IDs (the contact plan shows 11 contacts against 5 for the static one over time).

In other words, a static network graph grows depending on two factors: the number of nodes multiplied by the average egress contacts from a node. In comparison, a DTN graph

grows depending on three factors by adding the time component (multiplying the edges by the average number of subsequent contacts between two nodes before the contact plan horizon).

This pressure on the routing algorithm is problematic, as scalability requires higher node and contact counts, and flexibility and robustness require a later contact plan horizon. However, a relatively low ceiling for scalability can be expected due to the current state of research in deterministic pathfinding solutions while stacking support for different DTN aspects. Internetworking between regions operating with different contact plans can mitigate the issue but cannot be considered sufficient. Indeed, the risk is to transfer the problem from the intraregional level to the interregional routing one by increasing the interregional structure size and the possible scalability issues associated (e.g., delays and configuration).

The nature of such extreme networks might require redundancy as an efficient way to increase throughput and robustness between the regions. The administration might be, however, challenging, as both ends need to know about the possible options with potential optimization selections. At the same time, the delays within the region or between the two regions might be too high to allow reactive administration decisions. Configuration tensions may arise depending on the regional structure leveraged, for example, with the need for heterogeneous node configurations within a given region or contact plan sharing between agencies for gateway administration.

Routing affects volume management, as the suboptimal following hop selection can increase the risk of congestion, and reciprocally volume management affects routing as inaccurate volume booking awareness can lead to path selection encompassing contacts already exhausted in terms of volume.

Volume management is a critical concern, as the contact volume is precious, and incorrect volume management triggers a domino effect during high-load traffic periods, repeatedly updating the volume awareness incorrectly by relying on a state built on previous inaccurate updates.

A scalable solution associating accurate routing and volume management with robust and flexible internetworking does not exist for large scheduled DTNs.

1.3 Objectives

The main objective of this thesis is to improve DTN scalability by addressing three main topics: intraregional routing, volume management, and interregional routing. To provide some numbers, scalable in the context of this thesis means regions of hundreds of nodes and interregional structures of hundreds of regions. The state-of-the-art deterministic approaches are currently inapplicable with such network sizes. An algorithm aligned with the SABR expectations aims to schedule the bundles for routes that show early predicted arrival times and low hop count (in this order of priority). The objective of this thesis is to answer the following research question:

Can a routing algorithm increase scalability, network performance, and computational performance simultaneously, with respect to the state-of-the-art expectations, for both unicast and multicast bundles?

This question is decomposed into the following theses.

/T1/ An intraregional scheduled DTN routing algorithm for unicast and multicast bundles can show enhanced networking performance and reduced computational cost regarding the SABR standard expectations for the foreseen network sizes.

/T2/ The volume management can be processed accurately with a unified and efficient algorithm compatible with the support of priority and data rate variation extensions.

/T3/ The interregional routing for multiple actors can be supported with flexibility and reduced configuration pressure.

This thesis will introduce a set of new concepts that mark a shift in the current state-of-the-art pathfinding approaches, the current design of volume management, and the current design for interregional routing. A new routing paradigm involving shortest-path tree techniques for unicast and multicast purposes will be introduced, named the Shortest-Path tree approach for routing in Space Networks (SPSN), as well as a new volume management technique and an alternative regional structure.

1.4 Outline

This thesis is structured as follows: the objectives and requirements are covered in chapter 2. Chapter 3 covers the Delay-Tolerant Networks architecture and routing fundamentals and the required graph theory concepts for this thesis. Chapter 4 provides an extensive analysis of SABR. Chapter 5 covers the current state-of-the-art DTN routing approaches with special considerations regarding the requirements. Chapter 6 presents novel designs to address intraregional routing, volume management, and interregional routing support. The proposed designs will be evaluated in chapter 7. The core contributions will then be summarized in chapter 8, and an outline for future work will be proposed.

2 Requirements

This chapter proposes a use case scenario following the network structure predicted by the future Mars communication architecture report. A general requirement analysis is conducted, followed by the requirement formulations for each thesis presented in section 1.3.

2.1 Use cases

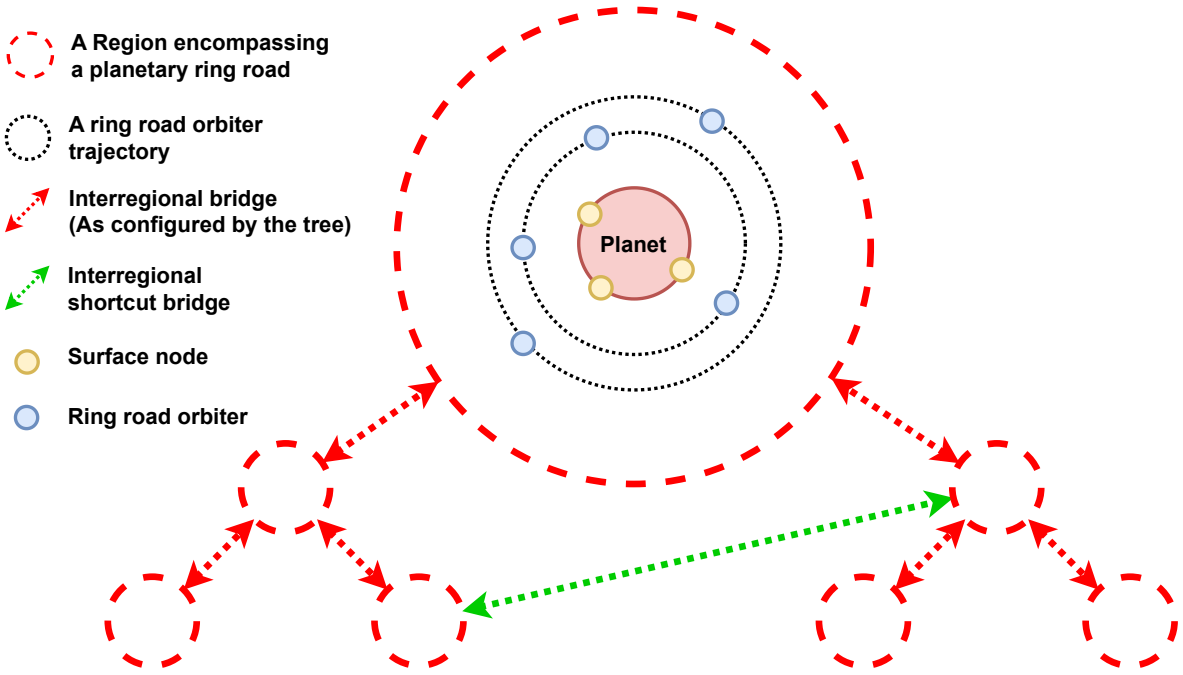


Figure 2.1: An interregional tree. The encompassed nodes of a region are represented for the root region only.

It has been proposed for the future interplanetary backbone to encompass relatively independent subnetworks called regions nested in a hierarchical tree structure [Ale19]. The passageway nodes would simultaneously be members of two regions and act as interfaces

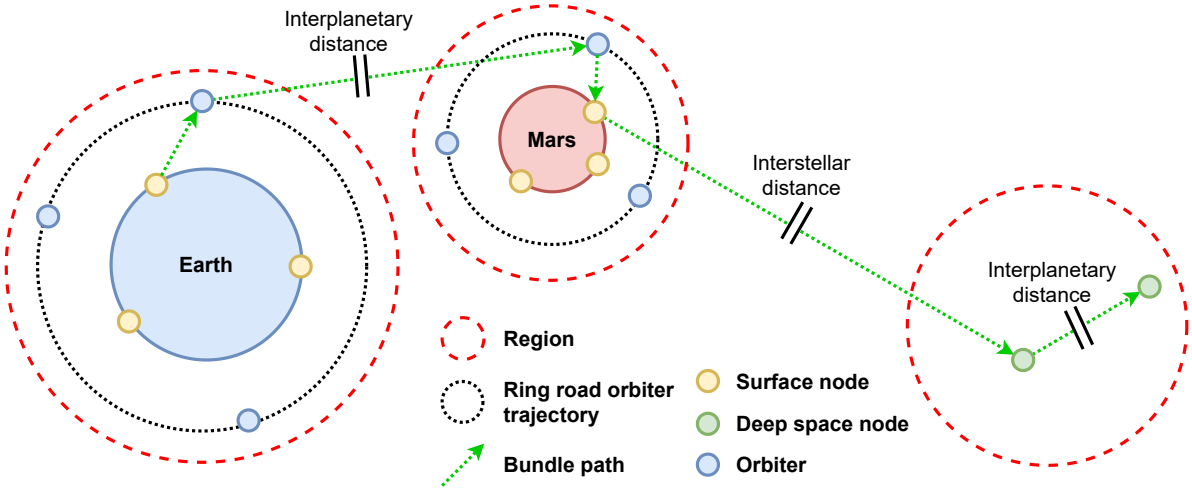


Figure 2.2: A deep space mission within the interregional networking backbone.

to allow interregional traffic. The interregional structure and forwarding mechanism will be covered in sections 3.1.6 and 4.7.

The scenario depicted in figure 2.1 will be leveraged to derive requirements for this thesis. The tree structure of the currently proposed interregional routing approach is retained. The figure presents seven regions in dashed-line red circles, but only the root region exhibits its intraregional topology.

The dotted red arrows depict the interregional bridges that are assured currently by the presence of passageway nodes, simultaneously members of the two concerned regions. Because the nature of those bridges is subject to changes, an abstract representation was intentionally preferred. The dotted green arrows depict possible bridges, called shortcuts, that would break the structure of the interregional tree. However, supporting such shortcuts could be highly beneficial to decrease the delays or increase the throughput.

The ring road network proposal, presented in [KBGB08], uses low Earth orbit (LEO) satellites to connect remote areas. A resource-constrained network with predictable orbit mobility patterns characterizes a ring road. The contacts can be very short and, therefore, can show limited volumes.

The ring road networks are convenient topologies for planet surface access within an interplanetary network. A single orbiting satellite can allow communications between DTN-capable nodes in space and on large areas of the planet's surface. Although not directly mentioned in the IOAG report for future Mars communications [ioa22], orbiting relay nodes communicating with surface assets in a delay-tolerant manner also constitute a ring road network. Ring road networks can be deployed for the moon [FFW18] and probably for any future exploration of a distant planet due to its cost-efficiency.

More specifically, their deployment within the regional routing backbone could also positively or negatively impact deep space missions. Communications are affected by ranges, atmospheric conditions, planet positions, planet rotations, and the hardware and energy required for transmission.

The atmosphere of Mars is, for example, 100 times less dense than the terrestrial atmosphere [Bar74], and atmospheric perturbations are expected to be reduced if compared with Earth. Ground stations are usually expected to be less impacted by constrained resource concerns. Moreover, the position of the possible intermediary planets and their nodes im-

pacts pathfinding.

Bundle end-to-end transmissions over interregional paths, as depicted in figure 2.2, might present non-negligible benefits to allow earlier transmissions or communications with reduced noise due to atmospheric conditions.

2.2 Requirements

2.2.1 Requirement analysis

The definition of functional requirements can be considered sensitive for this thesis. The route base properties required to select a route as the best candidate are already known and standardized by SABR, any route selection algorithm should align with the SABR requirements. Therefore, the approach shall minimize the bundle arrival time and the path hop count in descending order of priority. If several routes are still eligible after this process, extra filtering using the expiration times can be applied.

In other words, the main information provided by the SABR specification is the method to select one candidate as the best route among a set of existing routes, consistently implemented as finding the best candidate among a set of known routes. Depending on the processing pressure mitigation techniques being deployed, this process can not systematically ensure that the best existing route is part of the previously identified routes and, by extension, that this best route can effectively be selected. This behavior will be described in the section 4.

SABR selection criteria are assumed to be appropriate for the scope of this thesis. However, the conceptual location of this selection phase, either using a list of pre-computed routes described by SABR or directly during a hypothetical enhanced pathfinding algorithm, should not be constrained by a standard if the resulting outputs appear identical.

It can be possible to follow the SABR core but not necessarily the form, as SABR aims to approach routing correctness. The means to achieve this goal will be considered an implementation detail in the context of SABR. Experience shows that strictly following the recommendations (including a short mention of Yen's algorithm) does not provide a reasonable compromise between computational pressure and network performance with larger topologies (as mentioned in [ABB+15, FDJB21, FMCF18, Wal20, WBW+16] and confirmed in section 7.3.

Regarding software complexity, various measurement techniques exist, but one can argue that programming complexity exhibits a subjective nature [MM82]. However, the same study states that *"individuals tend to evaluate the complexity of the programming task more on the size of the resultant program than on either the expended effort or the inherent complexity of the task"*.

In comparison, the algorithmic complexity has visible and measurable effects. CGR is sometimes considered a complicated algorithm when diving into the details, justifying the publication of surveys and tutorials [ABB+15, FDJB21].

Lowering the software complexity is critical as reducing the algorithmic complexity lowers the risk of introducing bugs, simplifies the onboarding of new software developers and researchers, and eases the continuous enhancement of the software. These benefits are hardly measurable (objectively) and casually underestimated. However, support of new features can induce further software complexity.

2.2.2 Requirements relative to the routing algorithm

Scenarios from figures 2.1 and 2.2 highlight the routing challenges. The potential presence of resource-constrained nodes and the future network size predictions further constrain future approaches. The SABR standard is a baseline for the networking performance expectation as a constraint on the output. Concerning the input, the approach shall exhibit stable processing times for any bundle to be scheduled, and this at any time during operations. Multicast is a crucial feature for administration. Its support in ION's implementation of SABR, coupled with the network size increase prediction, motivates multicast support to be a requirement for this thesis.

Therefore, the requirements for thesis /T1/ are:

/R1/ Networking performance: The approach shall exhibit network performance comparable to the current SABR-compliant routing algorithms.

/R2/ Predictability: The approach shall exhibit a predictable processing time and memory pressure.

/R3/ Multicast: Multicast is an important aspect of administration and load optimization in a DTN. Multicast shall be supported natively.

Requirement /R1/ addresses the space networking needs, coupled with the node access of topological knowledge inherent to a scheduled DTN. The goal of the current state-of-the-art standard from CCSDS is hardly questionable. Deterministic pathfinding discards the need for replication. The path ordering from SABR is justified if early delivery, shorter routes in terms of hops, and use of longer life paths must be prioritized in that order. The use cases exhibit challenging topologies, with deep space communications and short scheduled contacts involving resource-constrained nodes. The solution provided in this thesis shall be efficient for the targeted region sizes, i.e., at least a hundred nodes.

Future space networks will encompass more nodes with better interconnectivity, meaning more interfaces and contacts. At the same time, all nodes might not have the same computational power, and requirement /R2/ intends to provide a routing algorithm that can show predictable processing time on a single bundle scheduling basis. The challenge of predictability in the context of DTN routing is to find the correct balance (which can vary depending on the node constraints) between pathfinding accuracy and path reusability to achieve the desired tradeoffs between the quality of the path detection and the computational pressure. The presence of resource-constrained nodes in the use cases brings the need to have controlled computational pressure.

Last, the algorithm shall be adapted with multicast /R3/ specificities. Indeed, CGR-multicast opened the path of multicast in the context of SABR, but the handling can be described as insufficient. Indeed, CGR-multicast uses CGR(-unicast) as a backend by computing and selecting paths for each multicast member. This approach lacks some effort sharing as the multicast group size impacts the multicast handling of pathfinding, best candidate routes selection, and volume updates. Concerning the latter, the volume handling for multicast is not SABR-compliant (see sections 4.5 and 5.4.2).

2.2.3 Requirements relative to the volume management

Contact volumes are precious resources, and the extreme nature of the scenarios from figures 2.1 and 2.2 showing short, long-range, or low data rate contacts (with possible com-

bination of those characteristics) renders volume management both crucial and challenging. Some level of local volume awareness accuracy is also a dependency for volume-aware routing algorithms like SABR. Bundle priority (see section 3.2.4) and data rate variations (see section 3.1.5) shall also be considered as elements of a general need to provide retro-compatibility, match the current state-of-the-art support, or prepare future support techniques.

The requirements for thesis /T2/ are:

/R4/ Accuracy: The main goal for volume management is an accurate internal representation of volume awareness.

/R5/ Compatibility: The volume management technique shall be compatible with extensions dealing with varying data rates and priority support.

/R6/ Adaptability: The volume management technique shall be adaptable to support potential future extensions like linear support of data rate variation (in opposition to the intervals of constant data rate solution).

Requirement /R4/ addresses the need for effective volume management, and a more accurate volume representation is welcome as a general solution to replace several existing features. Increasing the space network load (bundle count and sizes) will stress contact utilization management. Even minor inaccuracies for a single bundle scheduling can trigger a cascade of volume management inaccuracy for the following scheduling. As volume management directly impacts pathfinding, efficient volume accounting is crucial for space networks, including the considered use cases.

However, volume management is challenging due to other concerns like bundle priority, data rate variations, and the detection of rescheduling candidates. Simple approaches are easier to adapt to be compatible with those constraints. The base approach presented in the thesis mainly targets the latest bundle protocol version. Still, adaptability to older protocol versions or network-specific characteristics shall be possible. Requirement /R5/ shall ensure compatibility with those concerns. Even if priority support can be questionable in a multi-tenant network, deep-space missions can benefit from its support. Data rate variation is also a significant concern, as the bell shape of the data rate curves is a typical ring road ground-to-orbit communications characteristic [CFR18].

The approach shall also be compatible with prospective future designs. An attractive candidate is, for example, linear support of the data rate variation. The concept shall present adaptability for that possible enhancement by fulfilling requirement /R6/. Indeed, the high value of the space contact can motivate further optimization of the volume management for the considered scenarios.

2.2.4 Requirements relative to interregional routing

When operating on a regional structure as shown in scenarios from figures 2.1 and 2.2, the end-to-end paths may be longer in terms of hops and show higher delays, such characteristics increases the threats occasioned by possible single-hop failures (e.g., node breakdown or congestion). Redundancy is crucial to mitigate the risks by eliminating the single points of failure. Redundancy means the need for multiple simultaneous passageway support on a single interface level (between two regions). On a large scale, redundancy means the existence of multiple paths from one region to another (the tree becoming a mesh), at least to

allow better paths to exist if compared with a rigid tree structure. A passageway node operates with two contact plans (see 3.1.6). Requesting the nodes to operate with more than one contact plan is a severe concern for scalability.

The requirements for thesis /T3/ are:

/R7/ Robustness: Allowing multiple nodes to communicate simultaneously with the neighboring regions shall be supported.

/R8/ Flexibility: Shortcuts shall be supported to exhibit lower structural limitations in response to opportunistic possibilities and mission-specific needs.

/R9/ Partitioning: Clear partitioning is required to decrease configuration complexity and avoid potential scalability issues associated.

The main subject of questioning about the interregional routing approach was the single points of failure that constitute the node passageways. This issue is currently addressed in ION with failover support. The requirement /R7/ intends to support wider interfaces between regions (multiple simultaneous passageway support) with transparent management of such concepts. Moreover, large hierarchical regional designs are even more affected by single points of failure, bringing the risks of disconnecting large sets of nodes entirely upon failures of single passageways.

Interregional routing leverages a relatively static tree. The path from one region to another is unique, while interconnectivity could be possible between regions that are not supposed to be neighbors in the regional tree. The existence of possible shortcuts brings the need for flexibility with requirement /R8/. This requirement echoes the networking performance required, as shortcuts can also decrease congestion and delays for the considered use cases.

A more explicit regional border is essential for appropriately separating concerns and hardware between operators, fulfilling requirement /R9/. Discarding the need for nodes operating with multiple contact plans would also increase stability and scalability by lowering the responsibilities given to individual nodes, consequently lowering the number of routing tables to maintain.

3 Fundamentals

This section will mainly introduce the DTN field and the SABR [CCS19] functioning to provide the tools to understand any SABR analysis for the DTN routing context and the considered use cases. Moreover, this section will introduce the mapping between graph theory and DTN objects and the required graph theory tools to allow comparisons and support new concept descriptions. Lastly, a short discussion about determinism and predictability will be proposed.

3.1 Delay-/disruption-tolerant networking

3.1.1 Architecture

With the rise of wireless protocols and the miniaturization of computers, an increasing demand for communication with or between mobile devices has been observed within the last fifty years. Mobile Ad hoc NETworking (MANET) became a popular research topic in the nineties. At the same time, InterPlanetary Networking (IPN) foundations were independently developed to cope with the inherent delays and disruptions of deep-space communications.

Some IPN concepts were adapted in 2000 for challenging terrestrial networks showing disruptions and delays. The encompassing topologies sharing these characteristics were coined as *Delay-Tolerant Networking* (DTN) [Fal03]. Since 2007, RFC 4838 [CBH+07], 5050 [BFB22], and 9171 [SB07] specifications describe the DTN architecture and the bundle protocol.

The motivation for developing a DTN architecture was the inapplicability of existing network protocols showing such high delays and disruptions.

While the meaning of disruption is clear, no matter if the interruption of connectivity was planned or not, the cause of delay is twofold. End-to-end delays can occur if some links show disruptions. In some particular environments, the signal's propagation time from one node to another is not negligible. A prominent example is communication with deep space spacecraft, but the speed of light should not be considered the standard for signal propagation speed. Other environments are suitable for deploying DTN technologies, like underwater networks encompassing acoustic links.

The Internet protocols and applications leverage quasi-instantaneousness of communications, where disruptions lasting hours can routinely be observed in space networks. Protocols based on acknowledgments or conversations will likely fail to work correctly, as feedback

may be received hours later. This aspect presents two main consequences:

- The nodes along a path must store the messages until the next transmission opportunity. It represents an essential shift from the Internet world where the routers only forward the messages to a so-called *store-carry-forward* paradigm. It presupposes the necessity to provide storage to the routers. This aspect is a core component of the bundle protocol (RFC 5050 and 9171 [BFB22, SB07]).
- The application has to be delay-tolerant, and the DTN messages must provide enough data individually to allow the applications to progress when receiving a single message.

To this end, the bundle protocol adds the bundle layer, which lives between the application layer and the lower ones within the Open Systems Interconnection (OSI) model. Each protocol data unit, or bundle, can be arbitrarily long, making bundle sizes in the gigabyte range or higher possible.

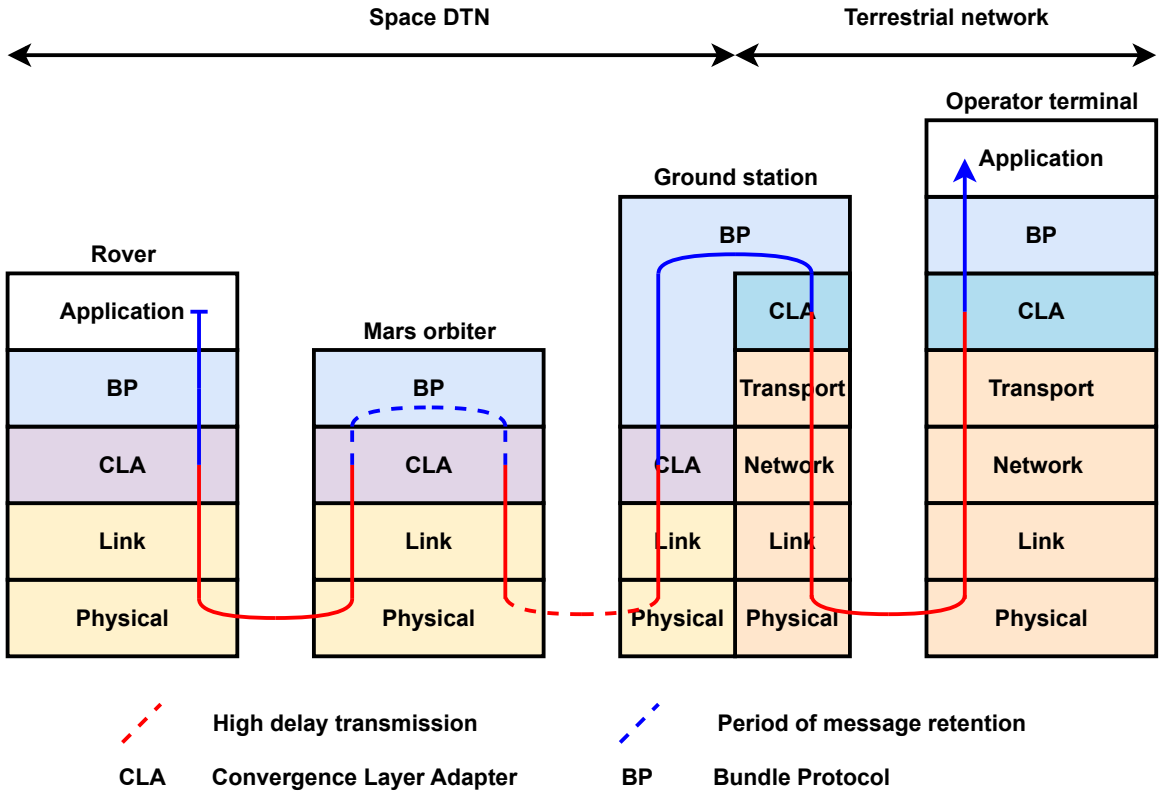


Figure 3.1: Bundle protocol example scenario.

To adapt to underlying protocols, the bundle agent uses Convergence Layer Adapters (CLA), which permit the transmission and reception of bundles via the lower layers. It makes the bundle protocol quite flexible, allowing agents, for example, to operate on top of TCP/IP for the Internet or the Licklider Transmission Protocol (LTP) [RBF08] for deep space networks.

Figure 3.1 depicts the end-to-end path of a large bundle (e.g., a video) sent from a rover on a distant planet to the Earth. An intermediary node is a satellite orbiting a distant planet that bridges the rover and the Earth's ground station communications. The communications between the ground station and the operator's terminal flows through the Internet. After the

video is recorded, a contact from the rover to the satellite is directly available, no retention is required, and the video is sent directly to the satellite within a single bundle. After transmission, however, there is no direct link from the satellite to the ground station on Earth. The message is stored on the satellite until the next transmission opportunity. The distance between the satellite and the ground station induces a very high delay on the link. The blue line depicts the layer locations where a bundle is still processed as a single data unit. The red line depicts the layer locations where the bundle can be processed as multiple distinct layer-specific data units.

3.1.2 Opportunistic and deterministic DTNs

As stated in the previous section, the bundle protocol applies to various challenging networks, from deep space environments, which brought the need to manage the delays on a single link due to the interplanetary ranges, to the mobile or vehicular networks showing non-deterministic motions and, thus hardly predictable disruptions. The contacts between the nodes can be predicted for the former, while future transmission opportunities are usually unpredictable for the latter.

At first glance, this can presume a clear separation between the deterministic DTNs, where a quasi full-knowledge of the future transmission opportunities can be leveraged with the opportunistic DTNs, where the next connectivity interval cannot be known in advance with sufficient confidence.

These two sets of DTNs are not nested but do overlap. A deterministic (or scheduled) DTN is not necessarily free of opportunistic behaviors. The DTN IP Neighbor Discovery (IPND) [EB10] permits the discovery of new participating nodes, and finally, in 2016, Burleigh et al. [BCMR16] introduced opportunistic behaviors within the deterministic routing approach of ION to finally leverage the information shared by the discovered nodes for routing.

3.1.3 DTN routing

With the delay and disruption constraints, the standard Internet protocols were not likely to work correctly in the bundle protocol layer. Some Internet protocols might work partially (e.g., the bundle protocol can be deployed on top of TCP/IP). However, multiplying long delays or disruptions along an end-to-end path affects those protocols negatively. Static router configuration can effectively provide relevant routing information at a given time. Still, the dynamic nature of the topology renders any static configuration ineffective most of the time or, at best, only intermittently acceptable.

More dynamic methods, such as distance vector routing protocols [Mal93, MM97, SNM+16], periodically update the routing table upon reception of update messages or detection of link state changes, are also ineffective in a DTN. The feedback of a topology change in a distant network area is not likely to be propagated in time to the local node for a correct routing decision for destinations depending on this topological change.

Routers using link-state routing protocols such as OSPF [FLM08] store a comprehensive network representation to apply pathfinding algorithms to update the routing table. In the same way, the internal network graph representation is updated upon reception of administrative messages, and even if the construction cost of the routes is negligible, the graph update is not likely to be performed in time.

Even if a collaborative protocol converges relatively rapidly on a terrestrial network, the end-to-end delays between the nodes and the frequent topology change far exceed the routing protocol capabilities to allow rapid convergence. These protocols are indeed designed to work in environments showing very sporadic topology changes over time, with end-to-end delays of the millisecond range, to rapidly find alternative paths within the second range. In opposition, the disruptions and single link delays of a DTN can routinely induce end-to-end in the minute or hour range, with contact durations that can be arbitrarily short.

Routing protocols targeting DTN have been developed (e.g., in [GFPBJ16, LYQ11, SRMS16]), including many proposals that can be included in various categories of the different proposed taxonomy designs. The large variety of routing protocols comes from the possible DTN topologies and node characteristics. A taxonomy will be proposed in the state-of-the-art section.

3.1.4 Contact plans

In a scheduled DTN, the periods of connectivity between the nodes, or contacts, are unidirectional and known in advance. It permits the creation of time-varying graphs representing the network and using a delay-tolerant version of shortest-path algorithms. The SABR standard [CCS19] also defines the information expected to be found in a contact plan. A contact plan is a list of entries gathering the necessary information to constitute a time-varying graph and to retrieve the signal propagation delays between two nodes at a given time.

In this document, the differentiation between prediction and observation is not necessary, as the pathfinding algorithms are unaware of the probability that a contact does not exhibit its predicted characteristics. Other definitions for the term *contact* are listed in [Wal20]. In the context of routing, the SABR standard defines a contact "*An interval during which it is expected that data will be transmitted by a sending node and that most or all of the transmitted data will be received by a receiving node*" (SABR, section 1.4.1, page 1-2). In other words, any bit of information transmitted during the contact will effectively reach the receiver if its prediction accuracy is high (i.e., the start and end time predictions match the observation).

A contact entry is described by (at least):

- Its start time.
- Its end time.
- Its transmitting node.
- Its receiving node.
- Its nominal data rate.

Therefore, a bidirectional contact requires two entries in the contact plan (from *A* to *B* and from *B* to *A*). Additionally, other metrics can be attached, such as contact confidence (a sort of informal probability) or CLA information.

This set of metrics does not reflect the possible delays on a single link. A contact can be long enough to exhibit different delays during its duration. It was therefore decided to create separate entries in the contact plans for this purpose.

Moreover, a contact is not a link representation, i.e., a contact plan written for SABR can not present two contacts with the same transmitting and receiving nodes with an overlapping period of connectivity. If it appears that two nodes will effectively be connected with

two different contacts that overlap in time, a third contact showing a data rate equal to the addition of those two contact data rates should be inserted for this overlapping period (and both initial contacts should be striped to end and start before and after this period).

Another contact plan entry type is used to retrieve the expected distance between two nodes for a given time interval to calculate the link delay. More specifically, SABR defines a range interval as *"a period of time during which the displacement between two nodes A and B is expected to vary by less than one light second from a stated anticipated distance"* (SABR, section 1.4.1, page 1-2).

A range interval entry is unidirectional and described by:

- Its start time.
- Its end time.
- Its A node.
- Its B node.
- The distance in light seconds between A and B. (One-Way Light Time or OWLT)

For simplification, some contact plans associate an expected link delay with the contact itself. It is the case with the simulator leveraged for evaluation (see section 7). If a contact was expected to show more than one delay during its lifetime, it could be split into as many contacts as range intervals relevant to the original contact.

3.1.5 Volume management

In the Internet

Reservation approaches for the Internet assume continuous end-to-end paths and are not likely to work with the frequent disruptions observed in a DTN.

The IntServ approach [BCS94] based on the Resource reSerVation Protocol (RSVP) (see [ZDE+93]) requires the receiver to request quality of service. This is a first critical issue as the delays on single links might not permit receiving the requests in time to allow enhanced volume management. The approach relies on the concept of flow, with router configuration along the end-to-end path, for a specific sender and receiver. The flows are also likely to be broken in a DTN due to disruptions, and the correction techniques would not allow continuous volume management support. Lastly, application-based flow control brings serious scalability concerns, as reservations are single flow-based rather than classes.

The DiffServ approach [NBBB98] relies on the concept of classes, and the class marking would be set at border nodes between subnetworks. This could be an issue, as fine-grained volume management is already required for intraregional networking. Assuming that the marking is available, the routers then apply scheduling and queue management policies, relying on the message classes.

The nature of the DTN field is unsuitable for existing Internet reservation techniques due to the abovementioned reasons, but also due to a paradigm shift from data rate concerns to volume concerns. On the Internet, the links are not expected to show disruptions, and working with data rates might seem sufficient to organize traffic control. In a DTN, the links are time-bounded intervals (with possibly long delays and low data rates). Knowing the volume transmittable via a single link seems more relevant than knowing the data rate of a contact duration without knowing its duration.

Contact utilization

This section defines how the contact data rate utilization is calculated. However, the principles presented in this section are not followed by the SABR specification. In SABR, the contact data rate R does not depend on time but is defined as the expected mean data rate over the contact duration D. The contact volume V is $V = R \times D$. For this reason, the data rate was termed *nominal* in the previous section. The contact plan does not inform about a contact data rate's variability or constancy.

The signal-to-noise ratio of orbiting satellites usually varies. It is at its maximum value when the satellite is at the zenith and its minimum value when it approaches the horizons. Given the Shannon-Hartley theorem [Sha49], which states that the theoretical data rate upper bound depends on the channel data rate and the signal-to-noise ratio, the expected data rate of contact may vary during its duration. This behavior of the data rate is known and already mentioned in [CFR18], and the associated curve depicting the date rate over time is expected to have a bell shape for transmissions between orbiting satellites and ground stations. A single contact can be fragmented into several contacts of different data rates to better fit the real expected data rate curve.

It becomes then possible to define a function f which gives for any time t between the contact start time A and end time B the corresponding data rate r. If transmission occurs during an interval $[a, b]$ ($a \in [A, B]$, $b \in [A, B]$), the transmitted volume v ($v \in [0, V]$) becomes:

$$\int_a^b f(t)\,dt = v \qquad (1)$$

Only the expected transmission end time b has to be calculated, as the volume v and the transmission start time a would be constants rather than variables: the transmitted volume is assumed to be equal to the bundle size, which is known, and a preliminary calculation would permit to set a before integral solving (e.g., a is set to the maximum between the contact start time and the current time, both being known). The expected transmission start time a would be known, thanks to some awareness of the previously booked volume for this contact, and obviously, the size v of the bundle is known as well.

A more convenient tool would be a function g derived from (1) returning for the transmission start time a, and the bundle size v, the expected bundle transmission end time b.

The DTN field moves towards accurately representing the data rate utilization [BCM+14, BTDT13, CDCP21]. However, each countermeasure against inaccurate volume booking assumes a constant data rate.

It is not expected to be a critical issue in practice, as the predicted contact volumes are usually accurate (a prerequisite for CGR's good functioning). This representation remains sufficient if the contact data rates are effectively used from the start time of the contacts. However, if some load is scheduled for a contact while the satellite approaches the horizon, the routing algorithm will assume a higher data rate than is actually available for this period.

From one extreme to another, if the load is a single large bundle carrying some precious science data, the contact will end before the bundle transmission end time, rendering the data rate interval utilization unprofitable (as the contact may end during its transmission). On the other hand, if the load shows plenty of small bundles, an arbitrarily high number of them can pass through, but in the same way, an arbitrarily high number of them may have to be rescheduled.

Even if using the hypothetical function g appears as the theoretically correct way to address this issue, the practicability of the approach seems, for the moment, unrealistic. Still, this can be considered for current volume management to support related future features.

3.1.6 Regions

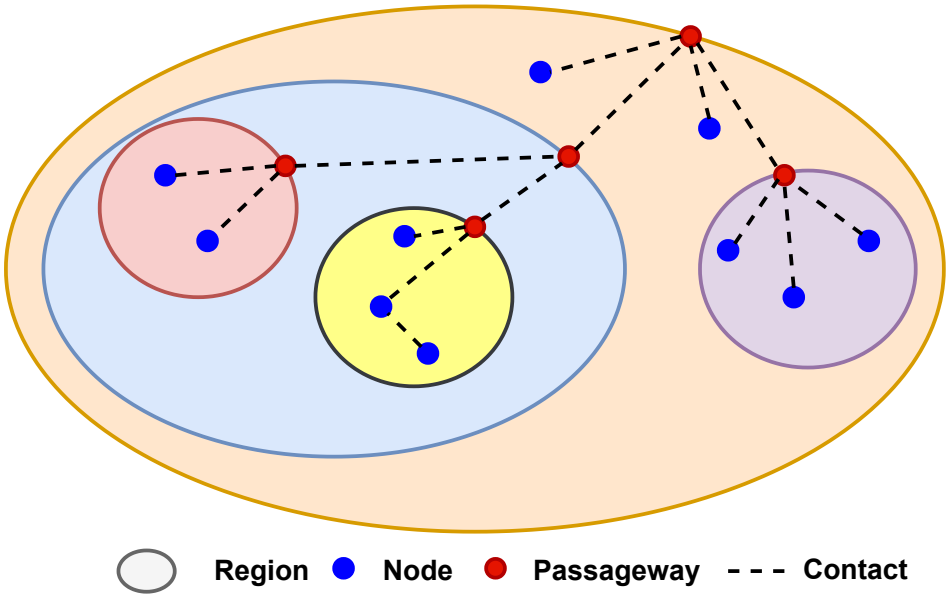


Figure 3.2: A DTN region hierarchy. Figure from [Jon19].

With increasing node counts, internetworking was adapted for DTN under the name of regions [Ale19]. The nodes encompassed in a contact plan are part of a region. The bundles are transmitted from one region to another through gateway nodes called passageways. A passageway is a member of the two regions it bridges. A passageway needs, therefore, the contact plan of both regions.

The approach is in continuous evolution, and this thesis considers a later version of the approach, as described in an unpublished document provided by Scott C. Burleigh [1].

As stated earlier, this permits a reduction of the contact plan sizes to minimize the computational impact of the routing algorithms within the region, as the pathfinding processing costs depend on the number of nodes cited in the contact plan (and the contact plan horizon depending on the implementation). Moreover, this can bring a separation of concerns to a certain degree, with the flexibility to create regions that are, e.g., topology-based, mission-based, or agency-based.

The regions are nested but do not overlap, as depicted in figure 3.2. The passageways are members of two regions simultaneously, the encompassing region and the nested one. In ION [Bur07], interregional routing is processed with a dedicated algorithm [Ale19] to detect which gateway shall forward a given interregional bundle. At the passageway bridging two regions, SABR is used on both sides. A region shall have a region ID superior to any lower regions accessible via this region.

If the interregional forwarding table does not carry an entry for a destination, the path is found with probes sent through the interregional tree. A passageway operating with a contact plan containing the target node would provide positive feedback, notifying all passageways along the reverse path that the target can be reached via those passageways recursively.

[1] "An Approach for Forwarding in an Interplanetary Network", June 2022.

The main difference between the version presented in [Ale19] and the earlier considered version is the probing, to shift to a discovery approach very similar to the approach proposed in section 6.4.3. Also, failover support is provided to replace a passageway node if a failure occurs.

3.2 Contact graph routing

3.2.1 A non-replication routing scheme

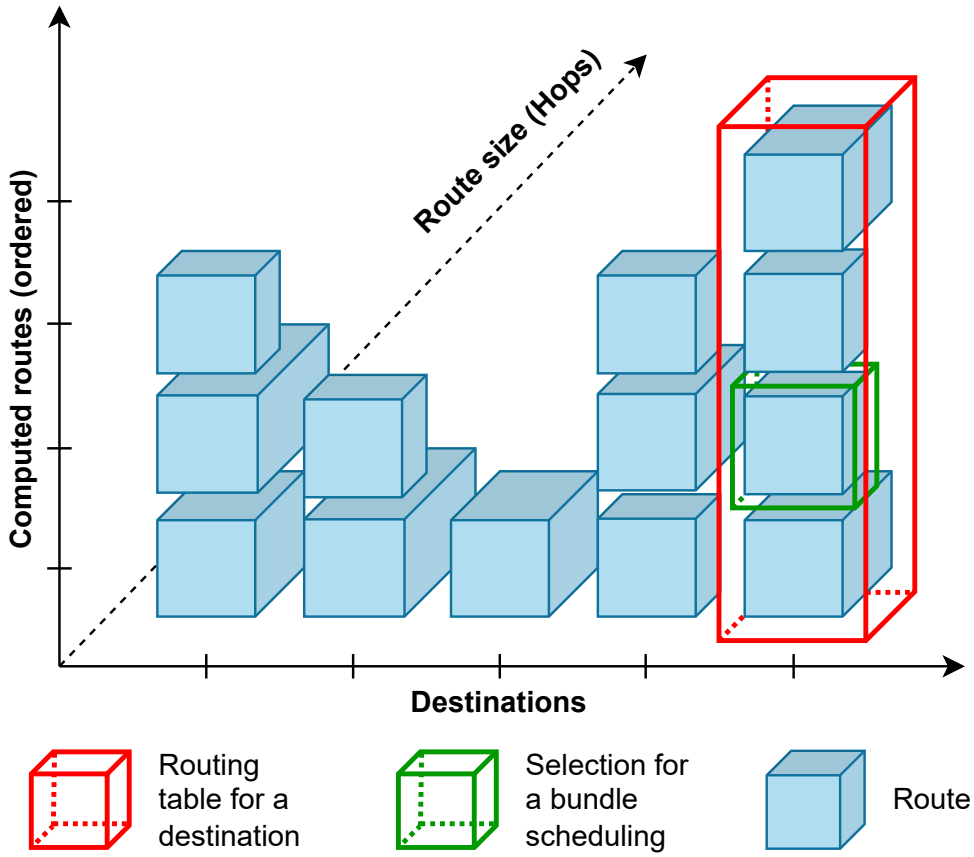


Figure 3.3: Representation of the route container expected by the SABR standard (observed in ION-CGR).

In a space DTN, a contact between two nodes has a duration, and an arbitrarily long period can separate this contact from the next one. Sending a bundle through a suboptimal contact (i.e., no paths are available from the receiver of this contact to the destination for an extended period) can drastically increase the bundle arrival time. In the same way, if the path is a dead-end, the bundle will never reach its destination, creating unnecessary load on the network and storage usage while decreasing the delivery rate.

Even if routing algorithms based on replication approaches [SPR05, VB+00] and probabilistic approaches [BGJ+06, LDDG12] may be deployed in scheduled DTN, contact predictability provides the possibility to deploy zero-replication using pathfinding methods.

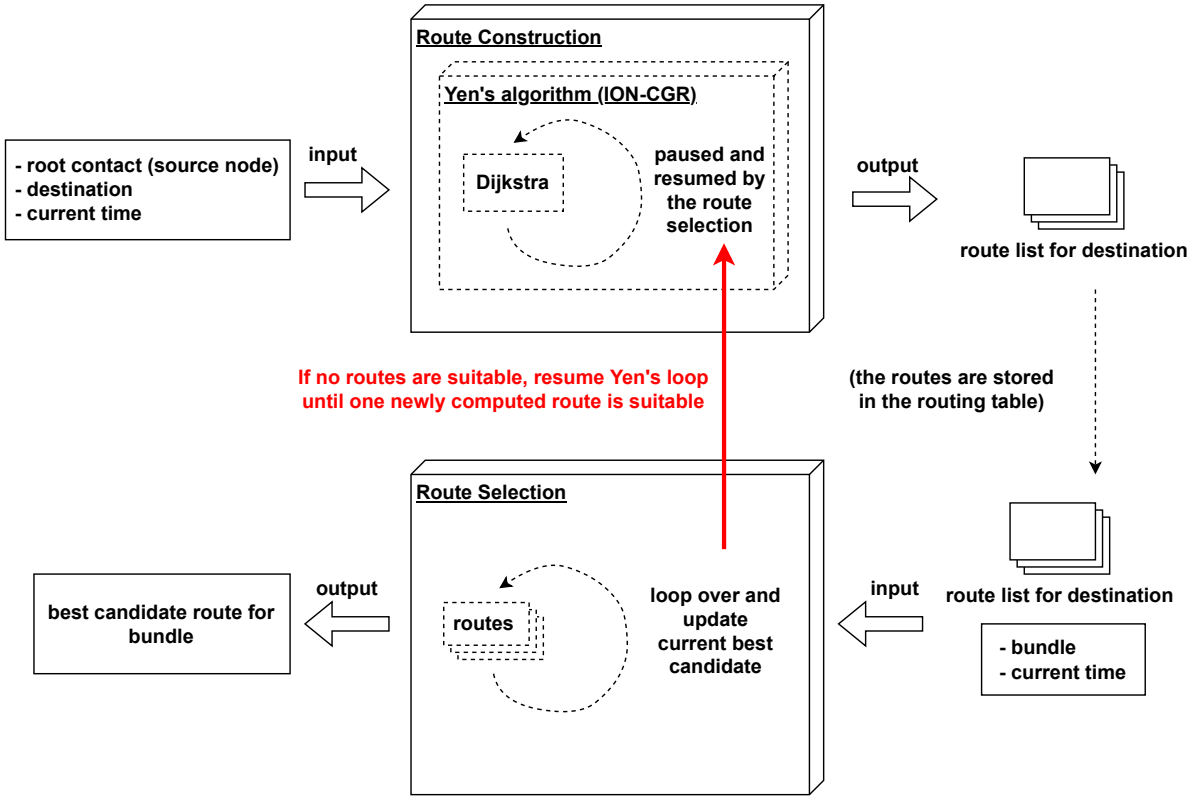


Figure 3.4: Simplified workflow of ION-CGR.

Contact graph routing [Bur09], which is the product of more than 15 years of development efforts, led to the SABR specification from the CCSDS [CCS19]. Indeed CGR has been productively used for communications with the International Space Station (ISS) [2] and is the sole routing algorithm available with ION in addition to static routing.

Using non-replication deterministic approaches based on Dijkstra's algorithm is appropriate for topologies that show almost no link uncertainties. Indeed, quasi-full knowledge of the network, resulting in high accuracy of the contact plan, permits CGR to provide excellent delivery probability and avoid congestion resulting from sub-optimal routing decisions based on probabilistic analysis or opportunistic behaviors.

At the scheduling phase, SABR proceeds to route selection. The route container structure can be represented in three dimensions as shown in figure 3.3. For each destination, several routes might be stored in the routing table. For completeness, a third axis is necessary to show that the routes are not equal in size as they present different hop counts (a hop corresponds to a transmission). Scheduling means a route (a sequence of specific contacts) shall be selected from the routing table for a particular destination.

CGR encompasses two phases. In the first phase, the route lists shall be populated, and the set of route lists constitutes the routing table. During the second phase, a route shall be selected in the route lists to forward a bundle. The SABR standard is precise in describing the second phase's steps but vague concerning the first, where much freedom is left to implementers. Yen's algorithm is, for example, only mentioned as a means to find alternative routes; its use is not mandatory.

[2] https://www.dlr.de/rb/en/desktopdefault.aspx/tabid-4769/5005_read-19087/

S. Burleigh first drafted CGR [Bur09] at the Jet Propulsion Laboratory (JPL). The CGR implementation within ION (version 4.1.0) is considered the reference for the following sections, and figure 3.4 depicts its workflow.

3.2.2 Route construction

This section will cover the implementation of the Dijkstra algorithm within ION (ION-CGR). The presentation of a specific implementation of CGR is crucial. As mentioned in the previous section, the algorithmic structure of the route construction is not constrained by the SABR specification.

The SABR specification, however, provides nomenclature for the metrics that are expected to be leveraged during route construction and the relation between them.

For example, the One-Way Light Time margin (OWLT margin) is defined "*as the maximum delta by which the OWLT between any pair of nodes can change during the time a bundle is in transit between them*" (SABR, section 2.4.2, page 2-5). The purpose of this margin is to take into account the distance variation between two nodes during the transit of a bundle between these two nodes and logically depends on the expected transmission delay between the two nodes, the speed of light, and the maximum anticipated distancing per second between two arbitrary nodes in the network (this is expected to be mission-specific).

The first byte arrival time metric depends on the OWLT margin: "*The first byte arrival time for a contact shall be the first byte transmission time for that contact plus the range in light-seconds from the contact's sending node to its receiving node, plus the applicable OWLT margin*" (SABR, 3.2.6.5, page 3-6). In other words, the expected delay plus the contact-specific calculated margin. It is assumed that a byte has a negligible transmission time.

The first byte arrival time can be considered the central metric within any SABR-compliant route construction algorithm. Most of the other metrics described, for example, the last byte transmission time, the last byte arrival time, and the projected bundle arrival time are obviously bundle specific and only calculated during a dry run (or delivery prospect) for each candidate route during the route selection. The concept of dry run is crucial for this thesis and is defined as follows:

A dry run is a simulation of the expected subsequent transmissions along the path, including the calculation of the transmission start and end times for each encompassed hop. The volume awareness can be considered but shall not be updated.

A valid formulation of the route construction functioning is that the algorithm searches a path for a virtual bundle of size 0 and a network without any load (volume awareness is not consulted).

Figure 3.5 depicts a simplified version of the route construction within ION-CGR. Each node in a DTN region is given a unique node number, and routing is processed from a source node addressed by such an ID to a destination addressed by another.

ION-CGR is said to operate on a contact graph, i.e., the contacts as vertices, to reduce the number of edges between two vertices to only one, to respect the original shortest path algorithm definition. This "contact graph" label is misleading and will be discussed in section 4.1. In the airline scheduling analogy, the vertices are flights, and the edges are layovers. It does not change the core principle of the algorithm. By definition, the distance to a target node in a contact graph is the first byte arrival time at a contact's receiving node if this receiving node is the target node.

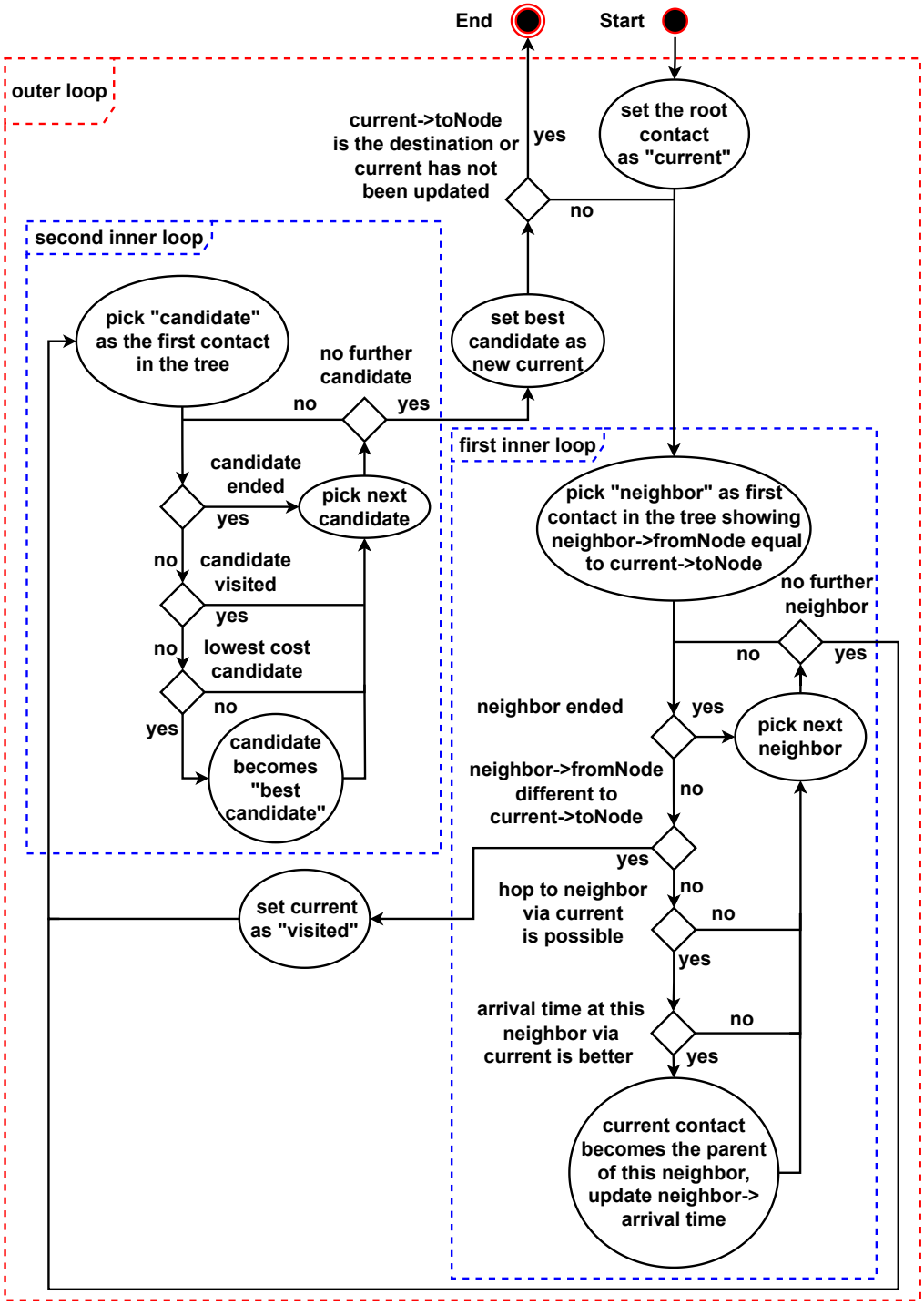


Figure 3.5: Simplified version of ION-CGR Dijkstra-based algorithm.

In ION, the contacts are stored in an ordered data structure, a red-black tree [GS78] with a hierarchical ordering. In the red-black tree, the contacts are first ordered by transmitting node numbers. The contacts having the same transmitting nodes are then sorted by receiving nodes. The contacts with the same transmitting and receiving nodes are sorted by contact start times. The tree permits placing the cursor at the first relevant contact and it-

erating over the next ones as long as these contacts remain in the appropriate range (e.g., contacts having the same transmitting nodes).

The search is processed with one outer loop and two inner loops:

1. The outer loop terminates as soon as the targeted destination is reached, i.e., the current contact receiving node is the destination, or as soon as there is no further contact suitable to be elected as the new current contact.

2. The first inner loop calculates the first byte arrival time at the direct neighbors from the current contact. A contact is a neighbor if its transmitting node is equal to the receiving node of the current contact. The current contact receiving node becomes the parent of each neighbor contact visited this way if the new distance to this neighbor contact is inferior to the former one.

3. The second inner loop then elects among the already-reached contacts and the new current contacts. This contact is defined as having the lowest first-byte arrival time (tie-break rules include hop and expiration checks in a SABR-compliant manner).

The reverse path can then be reconstructed easily from the final current contact by iteratively traversing the parent contact until the source is reached.

Exploration starts with a root contact as the current element. The root contact is a contact between the source node and itself.

Entanglement exists between the route computation and the route selection in ION (the route computation phase does not strictly precede the route selection phase). The alternative routes are found with Yen's algorithm but implemented adaptively. The routes are consequently not all computed beforehand but on demand. The consequences of this behavior will be covered in section 4.4.

3.2.3 Route selection

The routes calculated in the first phase are called *computed routes*. The second phase aims to find which of these routes are "candidate routes", i.e., they can convey the current bundle to the destination before the bundle's expiration time, and which of the candidate routes is the best and thus should be selected. Contrary to the route construction, a route selection algorithm can easily be derived from the SABR specification.

The route selection procedure iterates over the routes in the route list of the bundle's destination.

A dry run is processed for each route in a routing table to detect if this route can be a candidate. If the dry run fails, the route is discarded. Otherwise, the route can be a candidate, and the Projected Bundle Arrival Time (PBAT), the route hop count, and the route termination time are computed for this route. The termination time, or expiration, is the earliest contact end time among the contacts along this path, and the hop count is the number of contacts.

The dry run ensures that the bundle can effectively reach the destination using the considered route without encountering a contact with a *Effective Volume Limit* (EVL), or residual volume, lesser than the bundle size. The route can also be discarded at any hop if the bundle arrival time at an intermediary node exceeds the bundle expiration time.

The candidate routes can then be sorted by PBAT to elect the best candidate for forwarding. If the PBAT alone is not sufficient to compare two routes (same PBAT), the hop count

is considered (lower is better), and then the expiration time (later is better). If the application of those rules is still insufficient, the route having the smaller entry node ID is arbitrarily preferred. The entry node is the receiving node for the first contact of a route.

3.2.4 Enhancements and main features

CGR has been subject to improvements, with the three main domains of improvements being the earliest transmission opportunity, the priorities, and the overbooking management. The resulting proposals have been integrated into SABR and are presented in the following sections.

Earliest transmission opportunity and queue-delay

The current Earliest Transmission Opportunity (ETO) feature [BCT16], currently part of the SABR standard, is derived from an earlier version showing non-negligible differences (refer to [BTDT13] for the original ETO description).

This section describes the ETO functioning as described in SABR. It aims to adjust the first byte transmission time to an arbitrary route entry node by considering the bundles already enqueued for transmission to that node, pushing forward the first byte transmission time.

Because the bundles are not enqueued for a specific contact but for a specific node, i.e., the route entry node, the ETO is calculated for the cumulative duration of the contacts overlapped by the sum of the sizes of the previously enqueued bundles. The cumulative volume of the bundles enqueued for transmission to a given neighbor node can be greater than the cumulative volume of a couple of next contacts with this neighbor.

This approach has been extended to the next hops after the first by using the presumed volume previously allocated for those contacts instead of a cumulative volume enqueued, as the queue is accessible only locally. This extension is called *queue-delay* [CDCP21]. The queue-delay feature was arguably already present in [BCT16] (also named ETO for the contacts after the first hop). A non-negligible difference is that queue delay does not need to maintain an ETO for each contact in a contact plan, calculating it on demand during the SABR selection phase leveraging the volume awareness. While ION-CGR computes routes for abstract bundles of size zero for a network without any load, the queue-delay approach is more appropriate for real deployments.

Priority and overbooking management

In the previous section, the concept of priority was intentionally omitted to allow a smoother apprehension of the SABR and ION-CGR design. The SABR standard defines the priority as "*an indicator of the required order of precedence in the transmission of this bundle among other bundles*" (SABR, section 3.2.6.8.1, page 3-6).

SABR defines an EVL for each priority level instead of describing an "unscheduling" procedure to dequeue bundles of lower priorities to increase the contact EVLs for a higher priority bundle.

When a bundle has to be scheduled, the route selection only uses the residual EVL of the priority of the bundle. When a bundle of a given priority is scheduled, all the contacts along the selected path see their EVL for this priority reduced, as well as the EVLs associated with

the lower priorities. Depleting the EVLs associated with the lower priorities minimizes the risk of rescheduling too many bundles.

This approach can be understood as virtual channels, with each priority channel having access to a bandwidth shared with the lower priority channels.

The EVL management allows enqueuing of higher priority bundles to paths already congested by lower priority bundles as soon as the overbooking takes place [BCT16]. The overbooking management is meant to reschedule those lower-priority bundles. The SABR standard does not provide any recommendations and considers overbooking management an implementation matter.

3.3 Graph theory and DTN routing

3.3.1 Mapping with DTN objects

The representation of network nodes and links as vertices and edges of a graph is visually intuitive. With contact graph routing, vertices and edges are supposed to be contacts and retention periods, respectively. However, the practical difference is debatable according to the implementations. Indeed, whatever nature is given to a contact (edge or vertex), the exploration consistently anchors to nodes and uses contact hop metrics (edge weights) for distance calculation to reach neighboring nodes. Further analysis of the similarities will be provided in section 4.1, and for convenience, the vertices will be the nodes in this thesis unless otherwise specified.

The graph theory provides the tools to map DTN properties accurately. For instance, a DTN graph (the graph representation of a contact plan) is directed, i.e., the links are unidirectional to reflect the unidirectional nature of the DTN contacts. Two nodes may have several contacts connecting them. A graph showing several edges between two edges can be referred to as a multigraph.

The term multigraph can be considered deprecated or ambiguous, as stated in [W+01][3]. A less controversial language would be quiver. But since this thesis aims to tackle practical network problematics rather than focusing on graph theory, the term multigraph will conveniently be used to recall the possible presence of multiple edges (contacts) between two vertices.

Finally, a weighted graph is a graph that shows a single numerical value, the weight, assigned to each edge. Dijkstra's shortest path algorithm uses these weights to calculate the distance from the source vertex to an arbitrary destination vertex in the graph. Most of Dijkstra's algorithm implementations would assign a pair distance and parent (or predecessor) to each vertex the algorithm traverses during the search. At the initialization, the distances are set to infinity. From a current vertex for which the distance from the source has already been calculated, the algorithm checks if the distance of an adjacent vertex can be updated. A node reachable with a single contact (an adjacent vertex) is called a neighbor. If the distance assigned to the current vertex plus the weight of the edge connecting the current vertex to the adjacent vertex is inferior to the current adjacent vertex distance, the adjacent vertex

[3] A multigraph is casually defined as a graph that may have multi-edges but no self-loops [GY05, HR13], while others would allow both multiple edges between two vertices and self-loops [CLRS01]. In extreme cases like in [T+98], the term graph is used regardless of its structure but states that multigraph is a possible more general terminology, including graphs with at most one edge between a pair of nodes, but with possible self-loops.

distance can be updated, and the current vertex becomes its parent. The paths can then be retrieved from the reverse paths found by the algorithm.

The weights represent cost, lengths, or capacity. In a node graph, the cost of a DTN edge (a contact) is not necessarily atomic and is calculated from several metrics. The distance to a neighbor is then the return of a cost function, using as parameters the current node (transmitter) distance and the edge metrics (at least the start time and the delay are involved for this calculation).

If Dijkstra's algorithm considers a given bundle for the search, the distance to a neighbor (or bundle arrival time) shall be calculated using the edge data rate in addition to the edge delay. The edge volume (the product of the contact data rate and contact duration) shall also be calculated to check if the edge shows a sufficient residual volume for the bundle. If the edge volume is inferior to the bundle size, the cost function will return a distance equal to infinity. As described in the following section, this is separated into two phases as suggested by SABR and observed in ION-CGR. First, the routes are computed regardless of the load to schedule, and then the routes are tested to find the best candidate for a specific bundle.

3.3.2 Shortest path algorithm

Algorithm 1: Pseudo code of Dijkstra's algorithm's core when relying on priority queue to compute a shortest-path tree in a simple directed graph (maximum one directed edge between any pair of vertices).

Data: A simple directed weighted graph *G* map, (giving *G*[*vertex*]=*edge_to_adjacent*, *adjacent_vertex*), a source vertex *S*. Each vertex carries its distance. Each edge carries its cost.
Result: The parenting map *P* (giving *P*[*vertex*] = *parent*).

```
Initialize an empty map P
Initialize the priority queue accumulator Acc (lowest distance first)
Set distance of S to 0
Push the source vertex S in the accumulator
while True do
    if Acc is empty then
        break
    Get the lower distance vertex U from Acc
    foreach pair E, V in G[U] do
        Set D to distance of U + cost of E
        if V not in P then
            Set P[V] = U
            Set distance of V to D
            Push V in Acc
        else if distance of V > D then
            Set P[V] = U
            Set distance of V to D
            Push or update V in Acc
return P
```

Modern implementations of Dijkstra's algorithm computes a shortest-path tree using a

priority queue as discussed in [WC04], and a minimal implementation derived from this approach can be found in algorithm 1. Some comments will highlight the differences with the usual Dijkstra implementations within ION-CGR.

The weighted graph G is usually implemented as a hashmap. It sets the average complexity access of line 9 of the algorithm to $\mathcal{O}(1)$, where a red-black tree would show a complexity of $\mathcal{O}(log(N))$, N is the number of vertices, i.e., nodes or contacts. The performance of the hashmap depends on the function creating the hashes. If the hashes colid, the performance can significantly decrease.

Also, the use of a priority queue usually exhibits an insertion time of $\mathcal{O}(log(N))$ and a negligible access time to the lower priority item ($\mathcal{O}(1)$). ION-CGR does not need to insert in a priority queue because it usually does not use one. However, the selection of the next current contact (replacing the access line 8) exhibits a complexity of up to $\mathcal{O}(N)$, (N being the number of vertices, i.e., nodes or contacts). Therefore, even if the insertion time of a priority queue is discarded, access can be very costly (especially if the elements are contacts) and processed once per outer loop iteration (line 8).

Dijkstra's algorithm is designed to operate on a simple graph (at most one edge between a pair of vertices), and switching to a node graph design would inevitably bring a multigraph structure on the table, as subsequent contacts will exist in the contact plan between some pair of nodes.

This difficulty can be overcome in a very trivial manner with nested data structures[4].

It becomes clear that only the lowest cost edge of this list shall be used for the reverse-path construction. Complete proof that this approach is sufficient will not be extensively covered. Stating that a reverse path that does not use the lowest cost edge between two subsequent vertices on the reverse path is the best reverse path to a destination vertex would not hold very long against a proof by contradiction.

Adaptation to the DTN domain is also trivial. If the distance is defined as the earliest arrival time, and subsequent contacts between a pair of nodes cannot overlap (as constrained by SABR), then contact with an earlier start time has the lowest cost. Because the contacts are not likely to show dynamic characteristics, sorting those lists occurs at multigraph initialization only. It can be optimized by constraining the ordering of the contacts supplied by a contact plan, but this will be considered out of scope.

Consequently, if the contacts are sorted, the contact plan size (for a given node count) negligibly impacts the processing time.

It can not be considered as an original design, justifying its presence in the fundamentals section, as a similar multigraph structure was already employed independently in [Wal20]. A slight difference would be the priority queue design. The algorithm 1 suggests that the priority queue may be subject to an *update* operation (line 18) to change the priority of an object already in the queue (to change its position). This operation is not a standard, and many priority queue implementations do not provide it. When the operation is unavailable, just pushing a new element in the queue is sufficient. In this thesis, an original implementation that provides this operation is leveraged. However, this implementation is far from optimal, but the experience shows that the priority queue does not cause processing time issues. The priority queue design is considered out of scope and an optimization and implementation detail in the context of this thesis.

The main difference between a shortest-path tree construction and the original single-destination approach is the termination of the main loop. With the shortest-path tree, the

[4] In Python: *multigraph*[*current_node*][*candidate_neighbor*] would return the sorted list [*edge*_1, *edge*_2, *edge*_3].

algorithm stops when the priority queue is empty. The original algorithm stops when the destination vertex becomes the current vertex (line 8).

3.3.3 Edge and vertex contraction

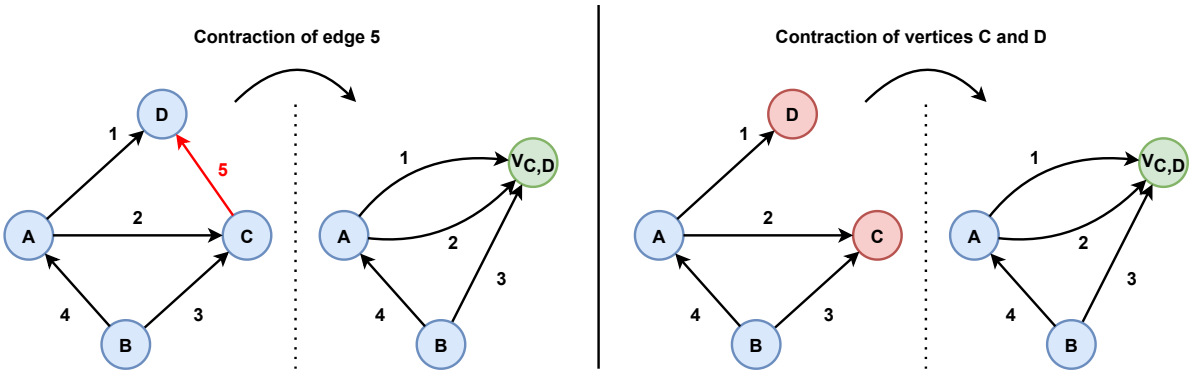


Figure 3.6: Edge and vertex contraction examples.

Edge contraction is a graph operation that merges two vertices by suppressing an edge showing those two vertices as endpoints (i.e., the vertices were adjacent thanks to this edge). This thesis describes the resulting node as "virtual" for convenience. This operation can reduce the graph size when an edge is identified as irrelevant to a specific problem. For example, an edge of cost zero in a graph leveraged for the shortest-path calculation can be a candidate for edge contraction, as this edge does not impact the distance calculation.

Node	Graph vertex
A	*A*
B	*B*
C	$V_{C,D}$
D	$V_{C,D}$

Contact	Virtual receiver	Real receiver
1	$V_{C,D}$	D
2	$V_{C,D}$	C
3	$V_{C,D}$	C

Table 3.1: Vertex mapping and contact fields the examples in figure 3.6.

Vertex contraction (or vertex identification) is a similar operation, but the constraint of selecting an edge for suppression is discarded. This allows the merge of non-adjacent vertices to simplify a graph in cases where vertices show similarities that can justify the merge. In the case of pathfinding in a DTN, vertex contraction can be leveraged if reaching a single node within a given subset is considered sufficient. This problem can be translated as finding the shortest path to a virtual node created after merging the nodes part of this subset.

Those operations can, however, induce information loss if no countermeasures are deployed. In figure 3.6, for example, the vertices *C* and *D* are merged. A first comment is that edge 5 itself is lost, but this also implies collateral losses. The resulting virtual vertex $V_{C,D}$ shows two issues. With DTN shortest-path calculation as an example, pathfinding to node *C* or *D* is still possible but requires some external mapping to inform that pathfinding to *C* or *D* should be translated to pathfinding to $V_{C,D}$.

Additionally, if such a virtual vertex is part of a path and is exactly the next hop from a local point of view, then information is missing to allow enqueueing to the correct transmission

queue, i.e., the node does not know if the bundle should be enqueued for *C* or *D*. This can also be addressed with mapping on the contact level, e.g., by attaching to a contact a "real" receiver to allow correct queue selection for transmission, while a "virtual" receiver is leveraged for pathfinding within the graph. Such mapping for the examples in figure 3.6 is proposed in table 3.1.

Another concern is the handling of contacts 1 and 2. If node *A* presents two interfaces, contacts 1 and 2 can overlap. After contraction, contacts 1 and 2 still overlap but now share the same receiver node, a configuration normally not allowed by SABR. Such overlapping can appear even if the contact plan (before contraction) is SABR-compliant. This requires either modification within the pathfinding algorithm to support such overlapping or contraction post-processing on the contacts to bring back SABR compliance.

3.4 Algorithmic determinism and predictability

An algorithm is deterministic if it always provides the same output for a specific input, in opposition to non-deterministic algorithms [BB96]. The fact that the program implementing such an algorithm will pass through the same states through the required control flow is also part of the deterministic nature. The time required to execute the algorithm depends on the underlying machine implementing the algorithm. However, the time needed to run the algorithm is also deterministic for a given input if no external perturbations are involved (e.g., the operating system scheduler).

Time execution prediction (or at least a trend) can be inferred from an algorithmic complexity analysis, the time complexity being one of its components. The other component is space complexity, leveraged to predict memory or disk usage. Therefore, if the time execution of a deterministic algorithm is known for a given input, the time execution will also be deterministic and equal to the first execution time for a second execution using the same input. The actual numbers for execution times, of course, depend on the hardware.

Taking Dijkstra's algorithm as an example, two executions with the same source and destination nodes would represent the same execution time because the algorithm is deterministic. If the source and destination input changes, but not the graph, the processing time is expected to be similar. The time execution variation might not be perceptible on a human time scale. Even if the input slightly changes, the deterministic components of Dijkstra's algorithm render it highly predictable as its time complexity depends on the graph input and not the source and destination inputs required for a single pathfinding iteration (the algorithm complexity of Dijkstra applies to any source and destination pair).

Mapping this discussion to DTN routing and CGR, the graph would be the network representation at a given time, and the source and destination input would be a bundle to schedule. The underlying functions of CGR are deterministic: as a result, the execution time would be the same for the same input (a graph representation with a given volume awareness and a given bundle to schedule at a specific time) while returning the same output (the routes computed and the candidate selected). Consequently, the memory and disk usage would also be the same.

A reliable DTN routing algorithm shall both be deterministic and predictable. Predictability would mean, for a given general input being the contact plan, that the execution time and memory usage pressure shall be similar when scheduling different bundles.

4 Preliminary analysis

The complexity of DTN routing, and the standardization of CGR with SABR for scheduled DTN, justifies a preliminary analysis to cover the intricated issues, known or newly discovered, that can arise when using CGR's pathfinding techniques. The border between concept design and implementation is blurry, as SABR does not constrain the design of the pathfinding algorithm for both the first path or the alternative paths to construct, rendering CGR's implementation within ION a reference material. SABR refers to the CCSDS standard, and ION-CGR refers to the implementation of CGR in ION [1]. CGR can refer to other implementations.

Although standardization is missing for pathfinding, the computational pressure and network performance of CGR are bound to implementation tools (e.g., hashmap for quick access in trees and priority queues for optimized Dijkstra implementation). Some of the issues are hardly coverable in a comprehensive manner without a short overlap between design descriptions and implementation examples. The introductory section 4.1 discusses the graph design natures before the functional analysis.

This section also covers the challenges of volume management and highlights the gaps that must be addressed in CGR.

Last but not least, the proposed approach for interregional routing (based on [Ale19]) will be discussed for analysis.

4.1 Node and contact graphs

The term "contact graph routing" stands for a routing technique leveraging a non-varying graph of contacts being vertices for pathfinding (and routing). The conducted analysis could not identify the need for a graph nature differentiation. Indeed, apart from implementation variations, operating with a contact graph data structure is conceptually indistinguishable from using a node multigraph.

The main reason to justify this differentiation is to get a list of contacts as the output. A route is a sequence of contacts rather than a sequence of nodes. Indeed, section 3.3.2 depicted a node parenting exploration technique for the reverse path rather than an edge (or contact) parenting exploration technique. This way, having contacts as vertices and processing exploration by updating a current contact as observed in ION-CGR (see section 4.3) seems appropriate.

[1] Version 4.1.0

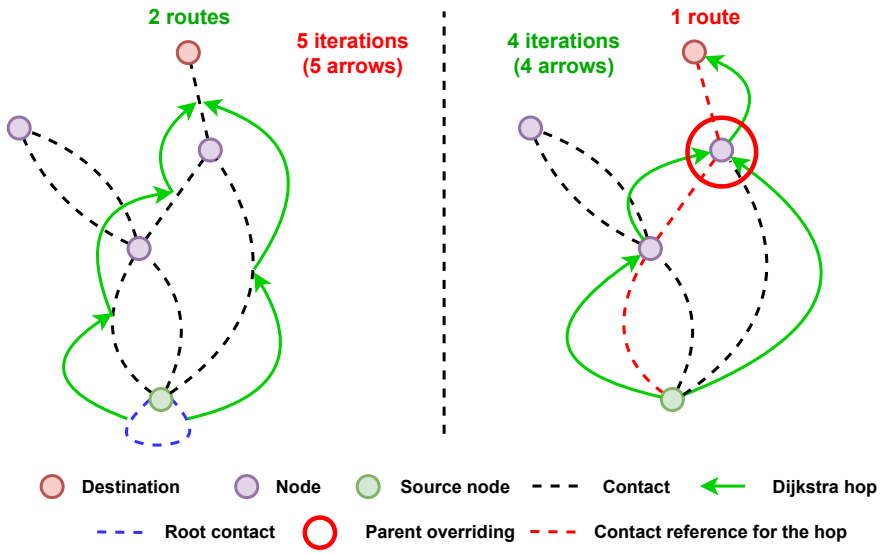


Figure 4.1: Difference in pathfinding effort and route detection when using contact parenting or node parenting.

However, pathfinding is still processed from a source node to a destination node. In the same way, the graph exploration in ION-CGR stops as soon as a destination node is reached (and not a destination contact). Even more explanatory, after the current vertex (contact) election, the neighbor contacts are contacts showing their transmitting node equal to the current contact receiving node, highlighting that the node can be considered the fixed point.

Other hints could inform about the conceptual nature of the graph. Firstly, a contact connecting a source with itself is an alias converting a node into a contact. This aliasing is necessary if the current vertex is (strictly) a contact rather than a node. Secondly, constructing a shortest-path tree as depicted in 3.3.2 seems meaningless if operating on a contact graph. Shortest-path tree connecting all contacts of the contact plan (including all the successive contacts between two given nodes) seems to show little practical uses. In contrast, a node shortest-path tree construction in a network clearly can show a purpose for routing (e.g., for multicast).

Finally, the contact graph data structure leveraged in ION-CGR is not a contact graph but a list encompassing all contacts (a red-black tree) with arguably efficient cursor positioning abilities. In comparison, an explicit mapping between a node graph and data structures is possible, as implemented in [Wal20].

This suggests a redefinition of the terminology to map the meaning of contact graph routing with the conclusions of this analysis: Contact Graph Routing (CGR) shall stand for a routing technique leveraging a time-varying representation of the network derived for a given contact plan for pathfinding.

A shift might be ongoing regarding the original contact graph design. Recent simulators like aiodtnsim [Wal20, WF19] and early-stage prototype implementations like [DJ19] are shifting to node multigraph representations, i.e., multiple edges representing contacts connect single pairs of vertices representing nodes.

Graph-wise, no difference could be detected. There is, however, a difference when varying the parenting strategy.

If the parenting is vertex-based and unique, a vertex (a node) has a single parent and a single "via contact" (the last hop contact used to reach the node), and they would be overridden each time a shorter distance path to this node is identified, as shown in figure 4.1. This could lead to suboptimal pathfinding results, but this can be considered rare and consequently highly preferable (due to processing improvement) according to [Wal20]. However, the negative effects of node parenting are expected to be more prominent if the topology becomes more complex.

Maintenance of a current element being a contact (as in ION-CGR) can prevent, in most cases, those issues and would not be incompatible with a multigraph data structure. However, other computational pressure issues would arise alongside other side effects (e.g., efficient support for shortest-path tree computation could be rendered more complicated). Those aspects will be covered in section 4.3.

4.2 Scenario

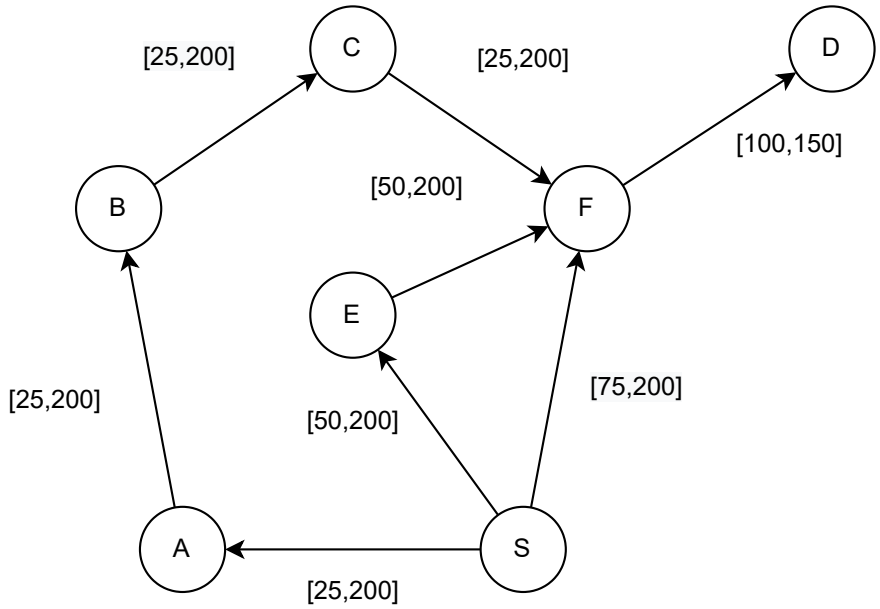


Figure 4.2: Example scenario to highlight path selection variation.

The scenario depicted in figure 4.2 will be considered for the following sections (contact format: [<start time>, <end time>]). The local node is node S, and each contact shows the same data rate. The links show no delays.

For a bundle of size zero having node *D* for the destination, the valid existing routes are:

- Route 1: $S \rightarrow A \rightarrow B \rightarrow C \rightarrow F \rightarrow D$ (arrival at $t = 100$ with 5 hops)
- Route 2: $S \rightarrow E \rightarrow F \rightarrow D$ (arrival at $t = 100$ with 3 hops)
- Route 3: $S \rightarrow F \rightarrow D$ (arrival at $t = 100$ with 2 hops)

As a reminder, the SABR standard defines the best route during the route selection as the route having the best arrival time. If two routes have the same arrival time, the route with

fewer hops shall be preferred. Therefore, the existing routes shall be ranked as follows (by distance, lower is better):

$$\text{Route 3} < \text{Route 2} < \text{Route 1}$$

4.3 Route construction in ION-CGR

This section will discuss the current route construction design in use within ION-CGR. Yen's algorithm [Yen71] is known to exhibit a high processing cost [Wal20]. The underlying Dijkstra algorithm design is also subject to weaknesses increasing the cost of a single pathfinding process.

The Dijkstra implementation of ION-CGR shows two subsequent loops nested in a main loop. The first nested loop updates the distances of the neighbor contacts from the current contact, and the second nested loop elects the next current contact. The main loop stops if the new current contact is "final" (the contact's receiver is the destination).

As briefly mentioned in section 4.1, ION uses a red-black tree to store the contacts. This data structure is accessed from both nested loops. Red-black tree capabilities were also mentioned in section 3.3.2.

Before entering the first loop, the red-black tree cursor is positioned at the first contact showing a transmitting node equal to the current node receiver. The loop iterates from the first neighbor contact pointed by the cursor and stops as soon as the transmitting node of the neighbor contact pointed by the cursor becomes different from the current node receiver. Two comments can be emitted from this design. Firstly, cursor positioning shows an average algorithmic complexity of $\mathcal{O}(log(C))$, C is the number of contacts in the contact plan. Secondly, having them sorted is only leveraged for the pre-loop cursor positioning. All the contacts showing the required transmitting node will be evaluated, even if most could be skipped. Subsequent contacts between 2 given nodes do not overlap (as constrained by SABR). Consequently, if two contacts between a pair of nodes can be used to reach the receiver node, the later contact (and any other later contact) can be ignored.

The second loop iterates over all entries in the red-black tree (the cursor is positioned at the first element) to elect the next current contact. The algorithmic complexity is $\mathcal{O}(C)$. In the original paper of Dijkstra's algorithm [Dij59], the use of a priority queue was already suggested by defining the set of vertices already reached as: "*the nodes for which the path minimum length from P is known; nodes will be added to this set in order of increasing minimum path length from node P*" with P being the source. The absence of a priority queue in ION can be explained by the fact that contacts are considered vertices (in reality, this means that the current element is a contact) and would therefore be inserted in the priority queue. As explained above, the first loop can iterate over many contacts, and many of them would consequently be inserted in the priority queue. The cumulative processing time of the insertions could increase significantly as a single insertion shows an algorithmic complexity of $\mathcal{O}(log(N))$, N is the number of contacts in the queue. While with a multigraph, even if the number of contacts is high (late horizon), the number of vertices can remain acceptable in regards to the priority queue (nodes are pushed into the queue, and not contacts).

This functioning can appear counterintuitive as a route length is bounded by the number of nodes in the network, regardless of the contact plan horizon, while the processing time depends on the contact plan size. A small thought experiment can further highlight this issue. Considering an imaginary network of 10 nodes operating with a contact plan with an

infinitely late horizon, a route from a source to a destination would encompass nine contacts in the worst case. Still, the computation of this route would last forever [2]

This could be the first reason that constrains ION-CGR to operate on short contact plans and have them updated more frequently (as, again, the contact plan size appears at least two times in the algorithmic complexity). Frequent updates can bring, however, negative consequences.

First, this puts pressure on the network operators, constraining them to design reduced-size contact plans, multiplying as well the contact plan update administrative bundles that have to reach their destinations for the good overall functioning of routing in the network.

Second, this implies that the routing table should be emptied to be repopulated with the routes calculated with the updated contact plan. SABR does not clearly describe this transition and this process can induce the deletion of suitable routes.

Concerning the pathfinding capability, the best existing route (in the sense of SABR) is likely to be found. At the same time, this statement does not hold when maintaining a current node instead of a current contact. This constraint is due to Dijkstra's algorithm limitations, with distances calculated from several metrics (which are discarded if maintaining a current contact rather than a current node). In figure 4.2, the first route found from S to D would be Route 1, as the earliest contact to reach F is $C \rightarrow F$. This contact is set as the taken contact of node F and C as the parent vertex. As covered in [Wal20] and mentioned in section 4.1, this behavior remains highly preferable regarding the computational pressure and networking performance tradeoffs.

4.4 Alternative route search

4.4.1 Yen's algorithm scalability

The current reference implementation of CGR in ION uses Yen's K-shortest path algorithm to search for alternative routes. Please refer to [FDJB21] for more information about the functioning of Yen's algorithm in ION. The argument for using Yen's algorithm is that CGR will effectively find the best routes for the route selection algorithm if the value of K is high enough. If a route's volume is exhausted, another route should be detected. Yen's algorithm can be used with both current element policies (contact or node).

The limitations of Yen's algorithm are not bound to a current element policy. The following analysis will consider a current node retainment.

Moreover, the adaptive implementation of Yen's algorithm will be considered as leveraged in ION. The main motivation is computational pressure reduction. It also presents memory pressure reduction and discards the challenge of defining the value K beforehand by increasing it dynamically. However, it does not prevent computational explosion, as covered in section 4.4.1. Yen's algorithm is paused as soon as a newly found route is suitable for a given bundle or resumed if a new route is required for a given bundle. It means that the routes available in the routing table would be systematically used until non of them is suitable during a bundle scheduling.

Considering figure 4.2, using a single parent node in Dijkstra's algorithm, the first path found to D would be route 1, because C would be the parent of F (arrival at $t = 25$). If a link

[2] The infinite list of contacts is not that absurd. The issue could occur if the list of contacts between 2 nodes is implemented as a generator leveraging an orbital mechanic prediction approach. Still, a finite contact plan is preferred.

of the sub-path $S \rightarrow A \rightarrow B \rightarrow C \rightarrow F$ is exhausted, a new Dijkstra search would find route 2 as the new best path from S to D because E would be the parent of F (arrival at $t = 50$). Route 3 would only be found at last.

To schedule a bundle to this destination, CGR iterates over the available routes and picks the best one. Given the order in which the routes are found, and the order in which the available routes in the route list would be selected, the scheduling of the bundle of this scenario would produce the following results[3]:

- If no alternative routes are requested, the bundle will reach node *D* via Route 1. A single Dijkstra call is necessary.
- If one alternative route is requested, the bundle will reach node *D* via Route 2. Five Dijkstra calls are necessary.
- If two alternative routes are requested, the bundle will reach node *D* via Route 3. Eight Dijkstra calls are necessary.

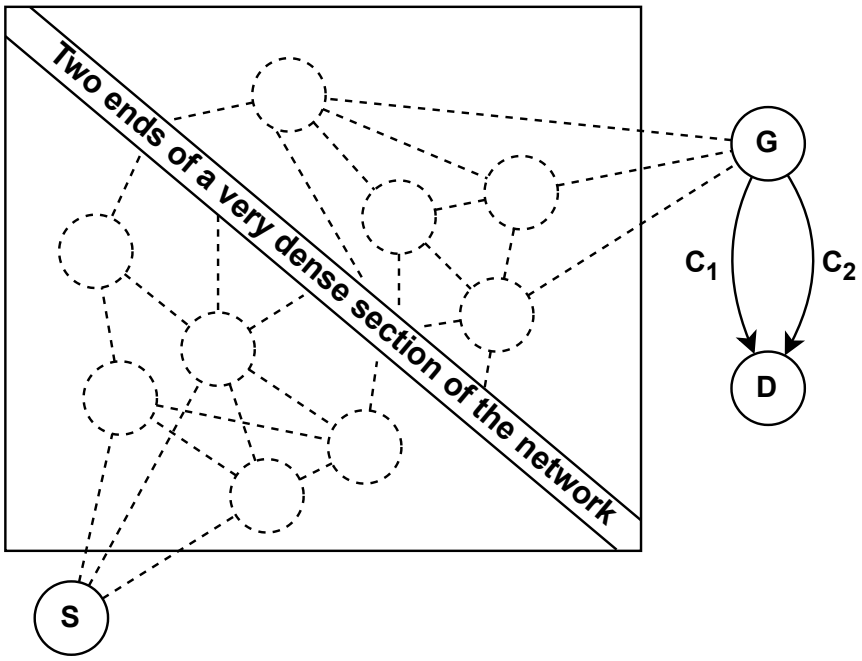
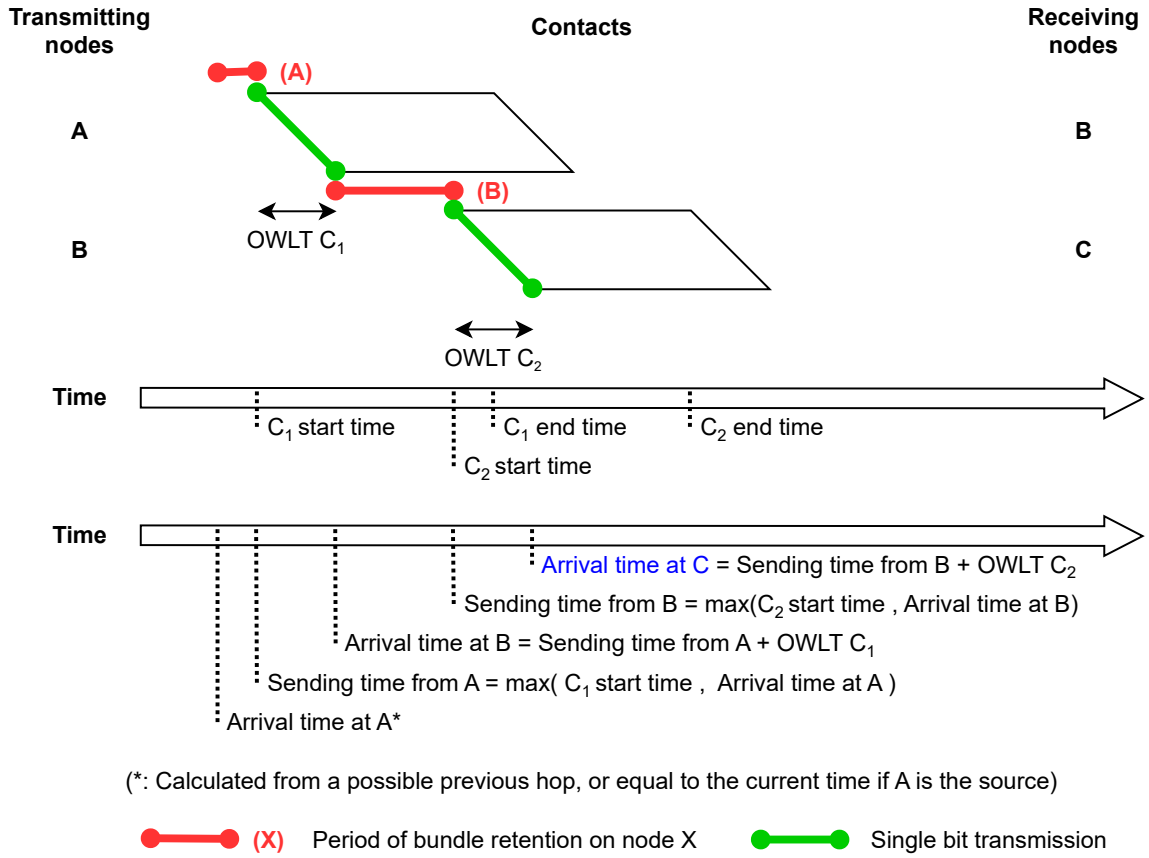


Figure 4.3: Network example that can trigger a computational explosion with Yen's algorithm.

Yen's algorithm shows three weaknesses. Firstly, the best existing route is not found by CGR if the value of K is not high enough. More specifically, this comes from the underlying Dijkstra algorithm leveraged by Yen's algorithm (here, by maintaining a current node rather than a current contact). It could be expected that the best existing route is effectively the first route found by Dijkstra's algorithm, but this is not necessarily the case depending on the implementation of Dijkstra (as covered in sections 4.1 and 4.3).

Secondly, the optimal value of K is load-dependent, which makes K impossible to define unless the expected load is known in advance. Even if the selected routes are not optimal,

[3]These results were confirmed by simulations performed with the aiodtnsim simulator.



Arrival time at C = Sending time from B + OWLT C_2
= max(C_2 start time , Arrival time at B) + OWLT C_2
= max(C_2 start time , (Sending time from A + OWLT C_1)) + OWLT C_2
= max(C_2 start time , (max(C_1 start time , Arrival time at A) + OWLT C_1)) + OWLT C_2
= max(C_2 start time , (C_1 start time + OWLT C_1)) + OWLT C_2
= C_2 start time + OWLT C_2

⇨ **The start time of contact C_2 is late enough to render the characteristics of the previous hops impactless. The distance calculation can be described as non-additive because the only relevant addition exhibits arguments that are characteristics of the same last hop contact.**

Figure 4.4: Example scenario highlighting non-additive behaviors during the distance calculation.

the delivery rate is not expected to decrease until congestion occurs. But if no alternative routes were requested, the load to node *D* would be scheduled on the contacts along route 1, reducing de facto the residual volumes of these contacts. This reduces the available contact volumes for bundles having nodes *A*, *B*, or *C* for destinations. If the value *K* is high enough, route 3 can be found and leveraged for the very first bundle that needs to be scheduled for *D*.

Thirdly, the complexity of Yen's algorithm makes it hardly scalable. The number of Dijkstra calls needed to find *K* routes is way higher than *K*. As shown before, even in a simple network

like the one depicted in figure 4.2, five calls are needed to find two routes and height to find three routes. To be more precise, the algorithmic complexity of Yen's algorithm depends on three factors:

- The complexity of the underlying Dijkstra algorithm.
- The number of paths to find.
- The average path length.

In a graph of N nodes and leaving aside the possible optimizations, Yen's algorithm exhibits a worst-case complexity of $\mathcal{O}(KN^3)$ with N^2 the complexity of Dijkstra's algorithm without optimizations [Yen71], and with a worst-case average path length being equal to N. This means that Yen's algorithm requires up to $K \times N$ Dijkstra's algorithm calls to find K routes in the worst-case.

Switching Dijkstra's current element policy from a node to a contact can be dramatic, as it drastically increases the processing time of the innermost loop of Yen's algorithm.

4.4.2 Blocking issues with Yen

Listing 4.1: Script leveraged to produce table 4.1.

```
def create_graph(size):
    names = list(string.ascii_lowercase)
    graph = {}
    edges = 1
    for i in range(0, size - 1):
        for j in range(0, size - 1):
            if i != j:
                if names[i] not in graph:
                    graph[names[i]] = []
                graph[names[i]].append(names[j])
                edges += 1
    graph[names[size-2]].append(names[size-1])
    return names[0], names[size-1], graph, edges

def rec_function(graph, curr_path, found_routes, destination):
    curr_node = curr_path[-1]
    iterations = 1
    for next_hop in graph[curr_node]:
        if next_hop not in curr_path:
            new_path = curr_path + [next_hop]
            if next_hop == destination:
                found_routes.append(new_path)
            else:
                iterations += rec_function(graph, new_path, found_routes, destination)
    return iterations

def test_with(size):
    found_routes = []
    source, destination, graph, edge_count = create_graph(size)
    start = process_time()
    rec_iteration = rec_function(graph, [source], found_routes, destination)
    stop = process_time()
    return edge_count, rec_iteration, found_routes, stop - start
```

As stated in the previous sections, Yen's algorithm is more likely to be implemented adaptively. This does not prevent problematic behaviors of Yen's algorithm, rendering it unusable depending on topological and traffic dynamics for relatively small networks (e.g., 30 nodes).

In the scenario depicted in figure 4.3, contact C_1 occurs before C_2, and they do not overlap. The following routes can be considered:

- The best route to node *D* shows contact C_1 as last hop contact.
- An arbitrarily high number of routes can exist with contact C_1 as the last hop contact.
- If contact C_1 becomes unsuitable (e.g., due to volume exhaustion), the next suitable route to *D* shows C_2 as the last hop contact.

While streaming bundles to node *D*, if a bundle scheduling renders C_1 unsuitable, Yen's algorithm's main loop must be resumed for the next bundle scheduling.

Because Yen's algorithm finds the route in order (by increasing distance in the sense of SABR), it becomes clear that all existing routes showing contact C_1 as the last hop contact need to be detected before detection of the first route replacing C_1 by C_2. Any routes showing C_1 as the last hop contact would be of shorter distance (earlier arrival time) if compared with any route showing C_2 as the last hop contact.

Finding the routes in order is the cause of this issue. Any optimization increasing the performance of Yen's algorithm cannot counterbalance the fact that numerous routes must be found due to the combinatory explosion. Consequently, any K-shortest path algorithm variant that finds the routes in the right order suffer from this issue.

In SABR, distance is casually calculated as if no addition were involved: the arrival time is often constrained by the last contact start time rather than an addition of the delays of the contact hops. SABR defines the best route as the route with the earliest arrival time and not the route with the shorter transit time (in opposition to pathfinding based, for example, on mileage, like pathfinding in a terrestrial road network). In a flight connections scenario, the arrival time of a passenger will be the arrival time of the last flight, regardless of the cumulative transit times and arrival times of the previous flights taken to reach the penultimate airport.

The algorithm description puts a lot of emphasis on the support of the interplanetary range delays or OWLT. If the OWLT of a contact is not null, the difference between the receiver and the transmitter distances will be at least the OWLT, but only on this single hop scale. The presence of an OWLT does not necessarily restore, on a path scale, an additive behavior to the distance calculation, as shown in figure 4.4.

This figure shows that non-additive behaviors can occur even with subsequent contact overlapping if, during construction, the arrival time at the first contact is before the second contact start time. As long as the transmission start time from C_1 plus its OWLT is inferior to the start time of C_2, the arrival time at node *C* of a bundle of size zero via this route (the null size being a CGR construction constraint) is dependent on the start time of C_2 only.

The first question would be to evaluate if this non-additive behavior of the distance calculation can be discarded. It implies that SABR's best route definition would need to be modified to casual prefer later arrival time routes. Earlier arrival times are preferable enough to be standardized by the CCSDS, and an efficient solution to fulfill this requirement shall show a higher priority. Discussions on best route redefinition are therefore out of the scope of this thesis.

A second question would be how often computed routes share the same arrival time and how impactful this behavior can be on the computational cost. For the latter, a strategy with simple rules can highlight how rapidly issues grow and how serious they are. The scenario will be, for a given node count *N*:

Vertices	Edges	Recursions	Routes	Processing time (secs)	Memory (bytes)
5	13	16	5	0.00001	528
6	21	65	16	0.00003	1800
7	31	326	65	0.00016	7808
8	43	1957	326	0.00112	41736
9	57	13700	1957	0.00962	266160
10	73	109601	13700	0.08090	1972808
11	91	986410	109601	0.87012	16659360
12	111	9864101	986410	10.17190	157825608
13	133	108505112	9864101	129.14977	1657168976

Table 4.1: Combinatory explosion results with increasing network sizes.

- Each node from node 1 to node $N-1$ is connected with the other nodes of this set with a bidirectional edge (two unidirectional edges), in a complete graph fashion.
- Node $N-1$ is connected with node N with a unidirectional edge E from $N-1$ to node N.
- Edge E is unsuitable due to volume capacity shortage, and it is assumed that this edge sets the arrival time of any existing route to node N.
- If a capacity shortage at edge E occurs, Yen's algorithm requires the computation of all the existing paths sharing this last hop edge before detection of another path using an alternative last hop edge, i.e. a second contact between node $N-1$ to node N, but with later start time (this second edge is not part of the graph for the tests).

The impact of Yen's behavior can be inferred empirically, thanks to the simple Python functions provided in listing 4.1. Yen's algorithm searches the routes in order until K is reached and, therefore, cannot leverage the same strategy. Indeed, additional logic is required to ensure that each found route is the n^{th} best route for each n^{th} iteration. This logic can be discarded if all routes are needed, as leveraged with this script. Thus, the results presented in table 4.1 can be considered optimistic regarding what can be expected with Yen's algorithm. The tests were conducted on a laptop with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz processor and 16 GB of RAM with the script functions of listing 4.1.

This table shall be understood as follows:

- Vertices: the total vertex count, including the last vertex N showing a single inbound edge named e.
- Edges: the total edge count, including the last inbound edge e.
- Recursions: the total recursive call count, the paths are computed by creating a longer loop-less path to each neighbor from the last hop of a current path by calling the pathfinding function recursively until the destination is reached. The metric gives an idea of the minimum iteration count to find all possible paths.
- Routes: the total route count from node 1 to node N. This highlights the combinatory explosion with an increasing number of edges.
- Processing time (secs): is the computational time required to detect the routes in seconds.

- Memory pressure (bytes): the memory required, in bytes, to store the routes. A route is a Python list of node IDs (being integers).

Please note that the support scenario excludes the time component of the DTN topologies. In a realistic scenario, many contacts can be encompassed between the local node's current time and the start time of a problematic last-hop contact. More than 150 contacts in a time-bounded subgraph between a current time and the start time of the first contact to reach a destination are not unrealistic, and the impact of this behavior had to be mitigated to allow evaluation in a realistic timeframe for realistic ring road scenarios (section 7).

Indeed, a "misplaced" contact in a ring-road scenario can trigger such a combinatory explosion. This issue was discovered precisely in these circumstances, with simulations using realistic ring road scenarios encompassing 30 nodes.

When the issue occurs, Yen's algorithm only stops once a route is found for the bundle that needs to be scheduled. This extreme behavior looks like a computational hang to an external observer and will be called this way for convenience.

The behavior is intrinsic to the nature of the algorithm and always present, while a hang will refer to cases where the behavior is not negligible anymore due to observable consequences. Defining a threshold is also complicated (e.g., a specific processing time or route count), from which the behavior shall now be described as a hang, as this threshold is probably hardware or mission specific.

However, some serious concerns can be emitted regarding the hangs due to the following characteristics:

- The risk of encountering a hang is hardly predictable, depending on the topology and the load.
- The processing time of a single iteration of Yen's algorithm can be predicted. But the duration of a hang is hardly predictable, as it encompasses an unpredictable number of Yen's algorithm iterations.
- A hang can postpone the scheduling of the next bundles enqueued for scheduling if the implementation is not multi-threaded.
- A hang can be damaging due to the memory pressure as the routes shall be stored. In table 4.1, the route count applies to Yen's algorithm container *A* but does not reflect the memory pressure of the intermediary container (Yen's *B* container). All those stored routes are unlikely to be used for routing, as they can all share the same discarding feature.

The memory issue is particularly concerning. It was initially planned to provide the table 4.1 for node counts up to 15, representing a total edge count of 183. But the simulation platform crashed at 14 nodes (157 edges) due to the memory pressure. The memory required to store the routes for the 13 nodes scenario has already reached 1.6 GB. Adding a few more nodes would already overwhelm the capabilities of modern-day simulation platforms.

The scalability issue of CGR, with the reference implementation using Yen's algorithm, is attributed to the algorithmic complexity of Yen's algorithm. This analysis is incomplete, as the number of iterations (i.e., increasing *K*) between two bundle scheduling is unpredictable. A single Yen's algorithm iteration shall not be considered an issue. However, a possible combinatory explosion requiring an arbitrarily high number of iterations for a single bundle is problematic. The scalability issue, i.e., hangs, is due to load and topological characteristics,

which are not reflected in the algorithmic complexity notation. If a single (or a couple of) Yen's algorithm iteration(s), i.e., a reduced number of Dijkstra calls, would be sufficient, Yen's algorithm scalability would be highly increased, as a single Dijkstra's call can be considered fast (at least while maintaining a current node rather than a current contact, more information in section 7).

Considering the statements made in section 3.4, CGR can hardly be considered as showing a predictable processing time and memory pressure when using Yen's algorithm for a single bundle scheduling.

4.4.3 Limiting contact approaches

To reduce the solicitation of Dijkstra's algorithm, the CGR limiting contact flavors were developed [FMCF18][4]. This approach reduces the number of Dijkstra's algorithm calls from KN to K. When a route is found, a contact of the path is elected as the limiting contact. This contact is then suppressed for the next Dijkstra algorithm call, providing de facto alternative routes (at least one contact is not shared with the previously computed route). This approach can also be implemented adaptively.

Constructing a new route with Yen's algorithm requires N Dijkstra calls (N being the average path length, bounded by the number of nodes in the worst case), while a single call is required when leveraging a limiting contact approach. The most interesting characteristic of the limiting contact approach is that it discards the risk of encountering a hang while still being able to compute many alternative routes.

The two policies to elect the limiting contact are first-depleted and first-ending.

First-ending

When using the first-ending approach, the limiting contact is the contact with the lowest transmission end time. The idea is to emulate time evolution by suppressing the contacts expiring first.

The route computation is then referred to as time-based. The problem with this approach is that several routes might share the limiting contact, and early suppression of such contact renders many routes undetectable. In the scenario depicted in figure 4.2, the first path found by Dijkstra's algorithm will be route 1. It appears that the limiting contact of this route is the contact $F \rightarrow D$, which is shared among the three existing routes. Having route 1 as the single route available in the route list of destination D shows the same limitations relative to pathfinding accuracy (the first found route might be suboptimal but still used until exhaustion if using an adaptive implementation).

First-depleted

When using the first-depleted approach, the limiting contact is the contact with the lowest available volume. The idea is to detect the first contact that would become depleted with continuous traffic to the destination. The route computation is then referred to as volume-based. But in the same way, the limiting contact might be shared by several routes, which would render suitable routes out of reach. And again, in the scenario depicted in figure 4.2, the contact $F \rightarrow D$ is the limiting contact, showing the lowest available volume.

[4] Limiting contact implementation available at `https://bitbucket.org/juanfraire/pycgr`

Issues

It can be argued that the heuristic leveraged for the election of a limiting contact is sufficient to compute new routes when required. If the contact volumes are not likely to be exhausted, the best route will be used for forwarding until the earliest contact expires, and suppressing this contact with the first-ending approach is sufficient to find the next best route. In the same way, if the contacts are short and their volumes are likely to be booked before the contact expirations, the first-depleted approach seems optimal.

However, the approaches become suboptimal if the limiting contact shall be excluded casually due to expiration and casually due to exhaustion (priority-dependent). Leveraging both methods at the same time could be an interesting option.

Another issue is suppressing a contact for a single destination, while a contact can be used to reach several destinations. If a contact is shared by two routes leading to destinations *A* and *B*, volume exhaustion of this shared contact for the route to destination *A* might trigger suppression of another contact for the construction of an alternative route to reach *B* (e.g., if using the first-ending approach).

4.5 CGR-multicast and shortest-path tree search

For a unicast bundle, the expected algorithmic complexity of the SABR route selection (no route computation involved) is at least $\mathcal{O}(RN)$, with R being the average number of available paths to a destination in the routing tables, and N being the path length.

For a multicast bundle, the complexity becomes $\mathcal{O}(RN^2)$. Moreover, the available route count per destination K can be very high in large networks. It is not unrealistic to observe $R > N$, if Yen's algorithm is used for alternative route construction.

The updates of the contacts' residual volumes along the chosen paths were left aside to calculate the route selection's complexity. If this aspect is taken into account, the algorithm should iterate over the contacts to update the residual volumes. This step is trivial for the unicast case, showing a complexity of $\mathcal{O}(N)$, but for the multicast case, one route per destination shall be taken into account, increasing the complexity to $\mathcal{O}(N^2)$, even with a marking mechanism to avoid multiple updates of a shared contact. The use of a shortest-path tree instead (see section 3.3.2) could simplify the process for the multicast case because the number of edges in a shortest-path tree is $N-1$, N being the number of reachable nodes (the number of edges can be higher if the tree carries the best existing routes in a SABR compliant manner).

The scalability issue can be expected, as the route selection now has a cost possibly higher than Dijkstra's algorithm. A legitimate hypothesis would be that leveraging a modified shortest-path tree search technique to get all the routes at once might be preferable instead of scanning each route list for each destination. Leveraging a shortest-path tree instead of route lists could be beneficial for a second reason, discarding the difficulties of the volume updates.

It shows some apparent benefits: if one path per destination is needed, the computational pressure is divided by the number of destinations. CGR uses a delay-tolerant version of the single destination search.

It is, however, complicated to integrate shortest-path tree concepts efficiently into CGR because of the single destination approach, handling of routes found for all destinations, and duplicate detection would also be required. Also, a "shortest-path tree version of Yen's algorithm" would be required, i.e., a delay-tolerant version of an algorithm specialized in finding the K shortest paths from a given source to each vertex in the graph.

In appearance, an attractive candidate for this purpose is Eppstein's algorithm [Epp98], which can find these paths with a compelling algorithmic complexity, but with cycles (or loops) of repeated vertices allowed. With a closer look, allowing loops can present a severe issue for the DTN case.

Again, the K-shortest paths are sorted in ascending order for the arrival times. Short-range communications are considered free of delays, which can leave infinite numbers of routes presenting loops. Each loopy route would be considered better than other potential loopless paths with higher arrival times.

Replacing Yen's algorithm or alternatives like Eppstein's requires further research.

As routes may stay valid for some time, storing routes in a routing table or caching a whole tree seem equally valid options. A conceptual sketch was presented in [Jon19] for the shortest-path tree construction and volume management, but again, an actual routing algorithm leveraging the prototypical back-end still needs to be presented.

4.6 Volume management

As mentioned in the previous sections, the volume of a given contact, or Effective Volume Limit (SABR, section 3.2.6.8.9, page 3-7), is equal to its data rate multiplied by its duration. This value is stored during the initialization phase of the contact graph. When a bundle is scheduled for a route, each contact along the path sees its volume reduced by the bundle size. Logically, when a bundle is transmitted, it uses the whole data rate for a given time.

However, the bundle transmission start time does not necessarily match the contact start time or the transmission end time of the previously scheduled bundle. If it was always the case, storing only the EVL for a contact would be sufficient. The actual representation of the volumes kind of stacks the cumulative bandwidth utilization at the beginning of the contact, even though complex networks are expected to create holes between the intervals of booked bandwidth.

This issue was already mentioned in [FDJB21, FMCF18]. Moreover, the envelope network scenario was presented in [Jon19] to highlight possible CGR routing failures caused by this volume representation, and an adapted version of this scenario is depicted in figure 4.5. Figures 4.5, 4.6, and 4.9 depict the same scenario and use the same coloring (two bundles, red and blue).

In this scenario, two bundles of size 5 are scheduled at $t = 0$ on the local node S for the destination D, all the contacts share the same data rate, and no variation can be observed during the course of the contacts. The first bundle to get a path for CGR is the red one and is scheduled for the route $S \rightarrow A \rightarrow C \rightarrow D$. The residual volume along the path is then updated. The second bundle, blue, receives the alternative route $S \rightarrow B \rightarrow C \rightarrow D$, because the contact $B \rightarrow C$ became depleted during the scheduling for the red bundle.

The conflict occurs on the contact $C \rightarrow D$, with CGR having two bundles to schedule at node C through the same link with insufficient available bandwidth, as depicted in figure 4.6. Consequently, the red bundle red is dropped[5].

The research field already attempted to solve this issue in two different iterations, with the ETO and the queue-delay features presented in section 3.2.4.

Even if the ETO feature addresses this issue, if the bundles are small, scaling bundle sizes will highlight some gaps in the approach.

[5]This behavior was confirmed empirically with simulations performed with the aiodtnsim simulator.

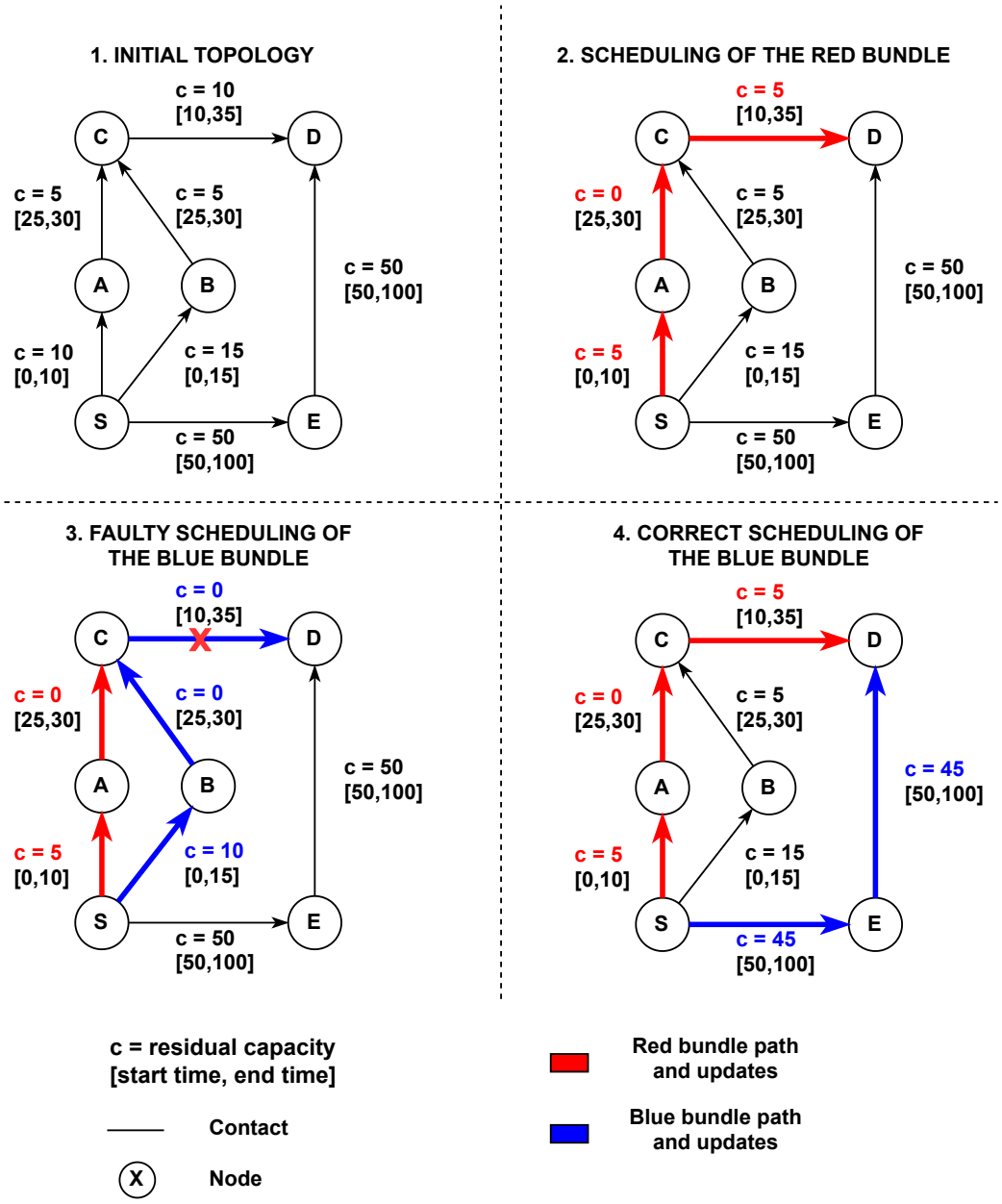


Figure 4.5: Two subsequent bundle scheduling from S to D. Figure adapted from [Jon19].

In the same way, the queue-delay addresses the issue to some extent but still shows some weaknesses. The queue-delay feature shows gaps in the presence of subsequent contact overlapping (see section 4.6.2, as the actual usage of specific data rate intervals, is not recorded.

4.6.1 Volume obstruction

The first issue, named volume obstruction in this document, occurs when the volume management of the routing algorithm does not recalculate an ETO at an intermediary node for the following bundles, as shown in figures 4.7 and 4.8.

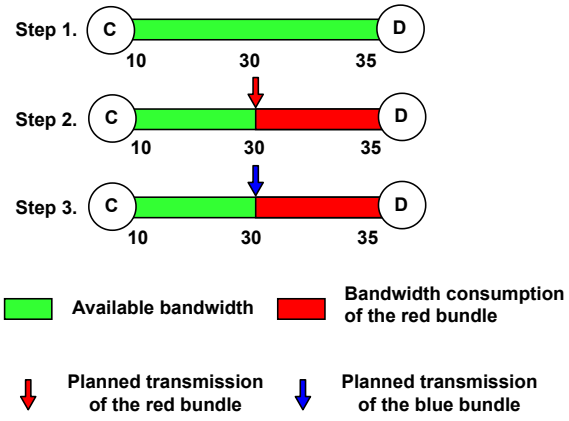


Figure 4.6: Data rate consumption for the contact $C \rightarrow D$ of figure 4.5. Figure adapted from [Jon19].

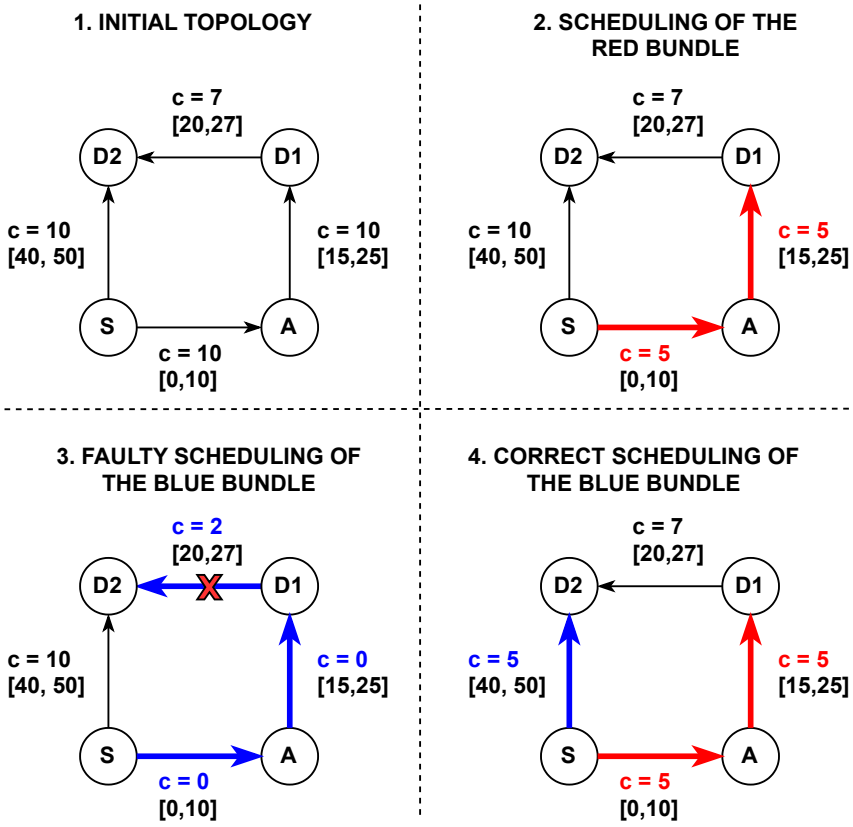


Figure 4.7: Scenario topology for figure 4.8.

The first bundle, identified as the red bundle, with $D1$ as the destination, is scheduled normally (top graph). Then when the blue bundle, with $D2$ as the destination, has to be scheduled, the ETO is correctly calculated for the first hop, but the bundle is then scheduled as if there is no volume scheduled on the next hops (middle graph).

The correct expected scheduling behavior would be to detect that the bundle *blue* cannot

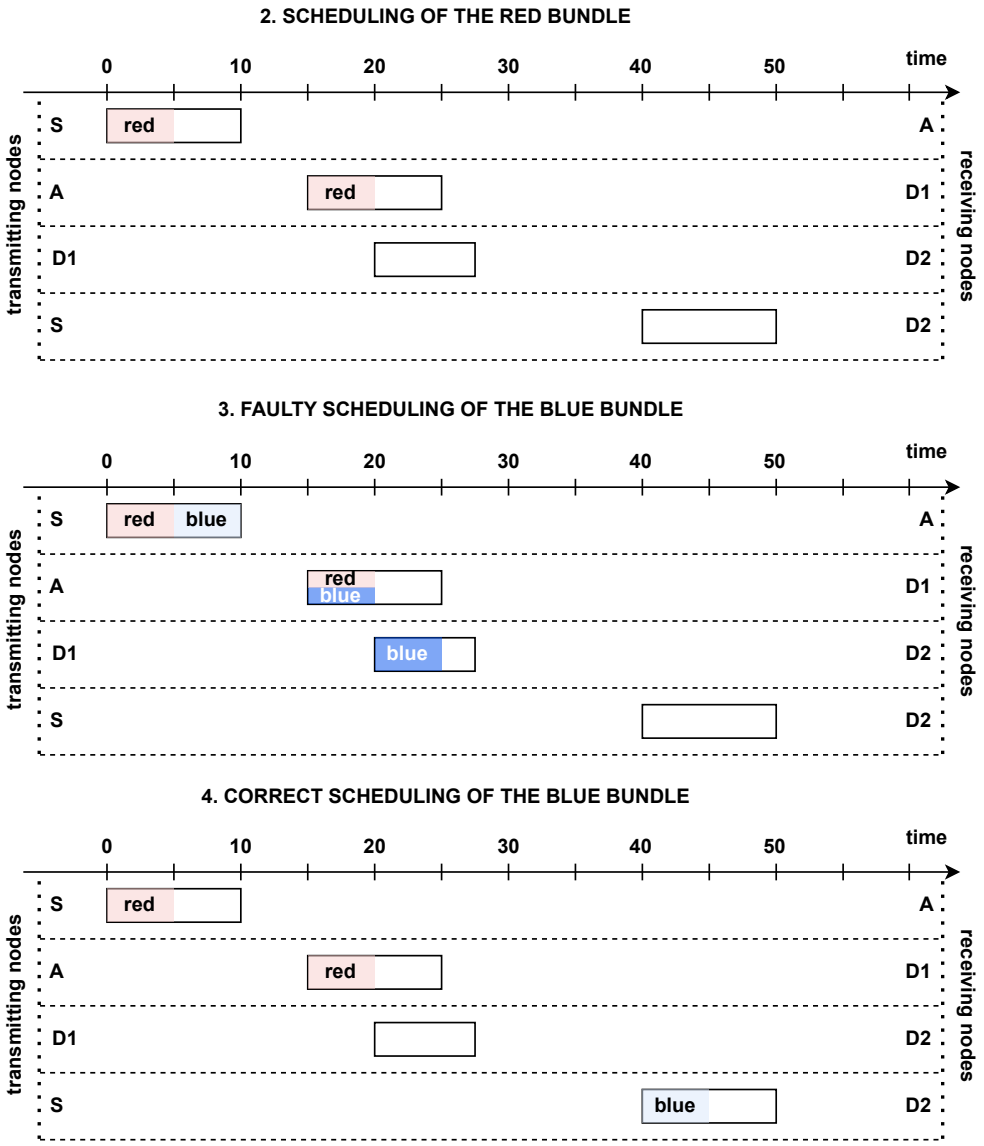


Figure 4.8: Volume obstruction issue scenario. The dark blue boxes depict faulty volume booking. Scenario topology is depicted in figure 4.7.

go through the path $S \rightarrow A \rightarrow D1 \rightarrow D2$ and should be transmitted through another path (bottom graph). In such a configuration, the queue-delay feature is sufficient to allow the selection of the right routes.

4.6.2 Contact sink

In opposition to the volume obstruction, the second issue named here is the contact sink behavior (the ETO falls "at the bottom of the sink" due to the early contact start time, i.e., the sink depth, instead of behind accurately set to a higher value). The issue occurs when a contact acts as a sink for the volume booking, rendering the ETO calculated by the queue-delay feature inaccurate. Figure 4.9 is another representation of the envelope network (figure

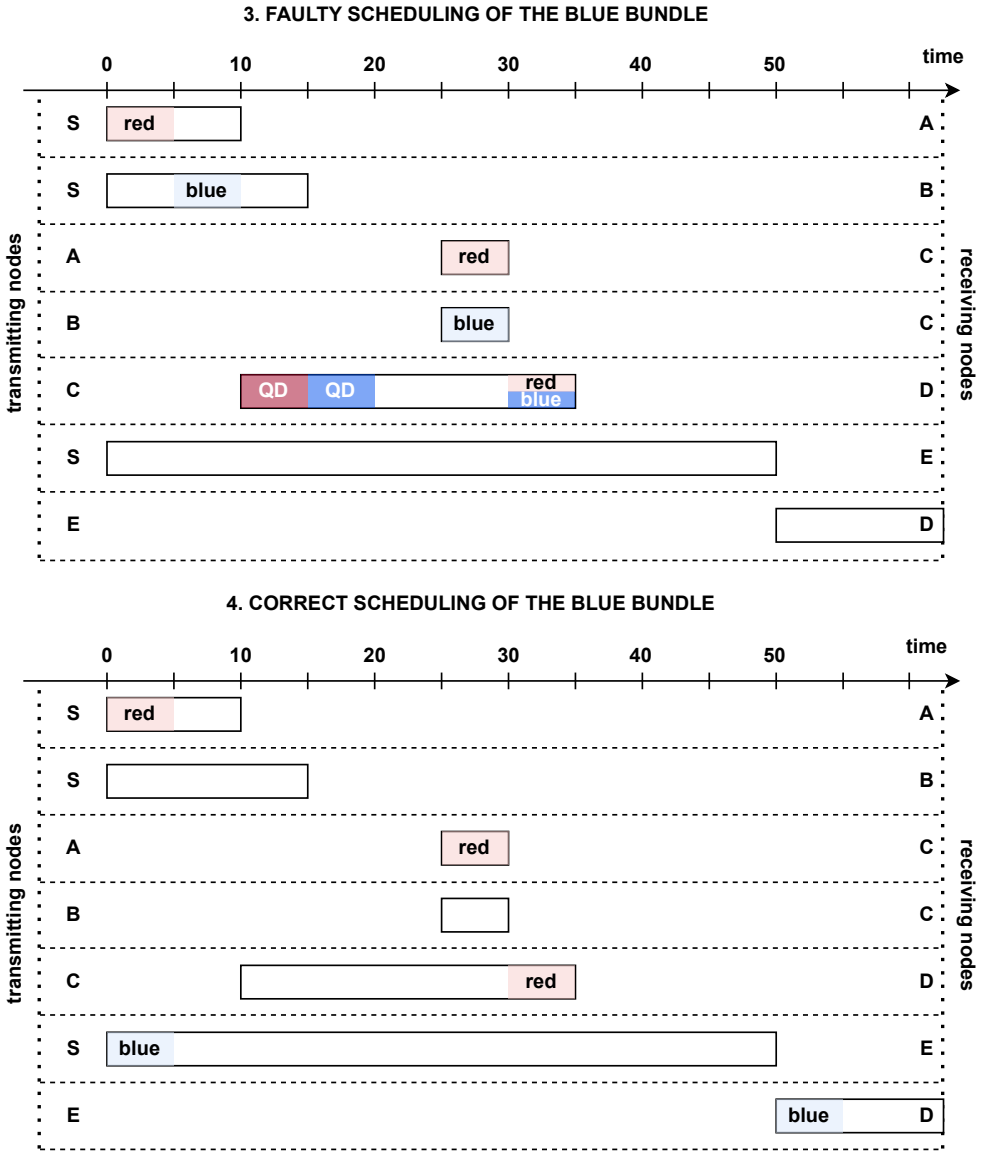


Figure 4.9: Contact sink issue scenario. The dark blue and red boxes depict faulty volume booking. Scenario topology is depicted in figure 4.5.

4.5).

The queue-delay calculation is applied at the start of an intermediary contact to get an earlier transmission start time by offsetting the contact start time with the expected cumulative transmission time for the already scheduled volume. But if the start time of the contact is earlier than the first bundle's real transmission start time, the "hole" is ignored.

Indeed, the queue-delay would always stack the transmission from the contact start time for each bundle scheduled. Again, a resilient routing algorithm would select an alternative path for the second bundle (bottom graph). In other words, overlapping is manageable by queue-delay to prevent volume obstruction, as long as the second contact does not act as a sink.

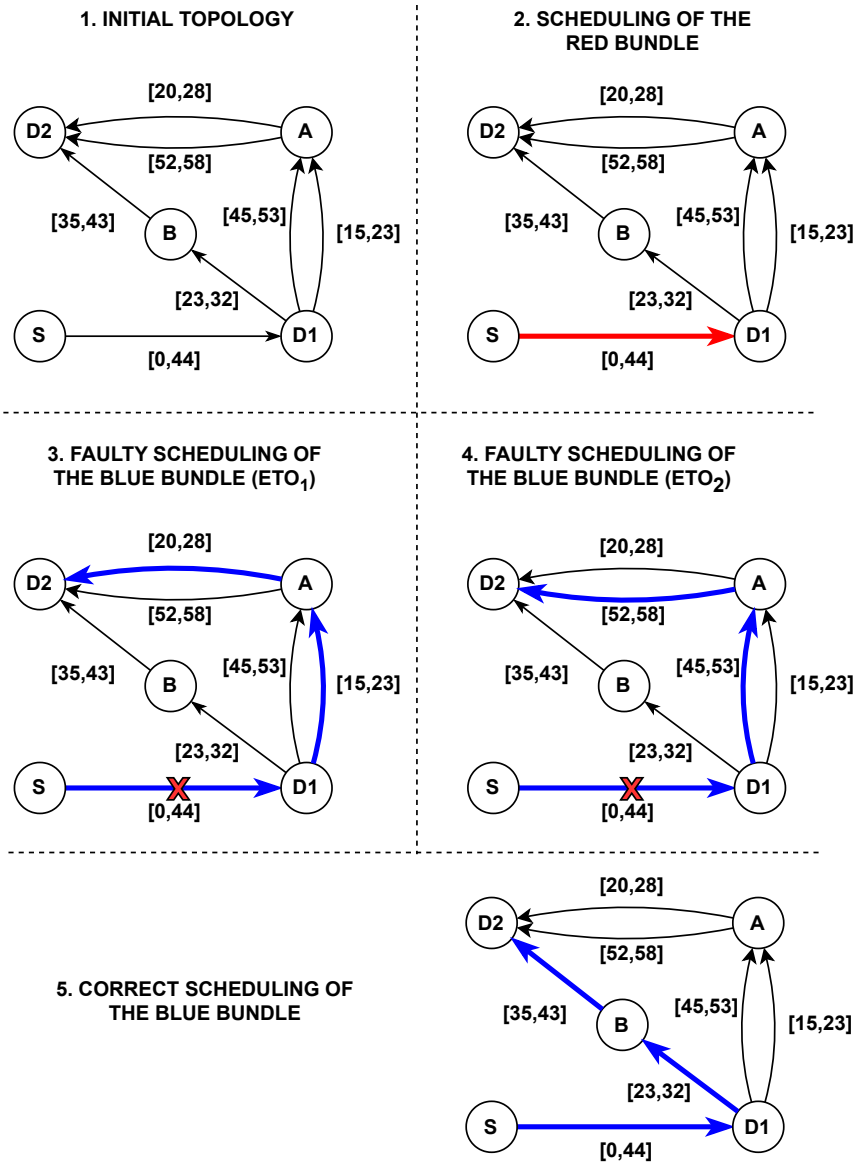


Figure 4.10: Scenario topology for figure 4.11.

4.6.3 Ghost queue

The last issue, named here the ghost queue issue, occurs when the blue bundle has to be scheduled during the transmission of another large bundle, here the red one, as shown in figures 4.10 and 4.11. The red bundle has for destination $D1$ and blue bundle has for destination $D2$. The subfigures show three different cases, and the scheduling occurs at t_0:

- The red bundle is considered out of the queue as soon as transmission starts (top graph). The ETO is then t_{eto1}.
- The red bundle is considered out of the queue only after its transmission termination (middle graph). The ETO is then t_{eto2}.

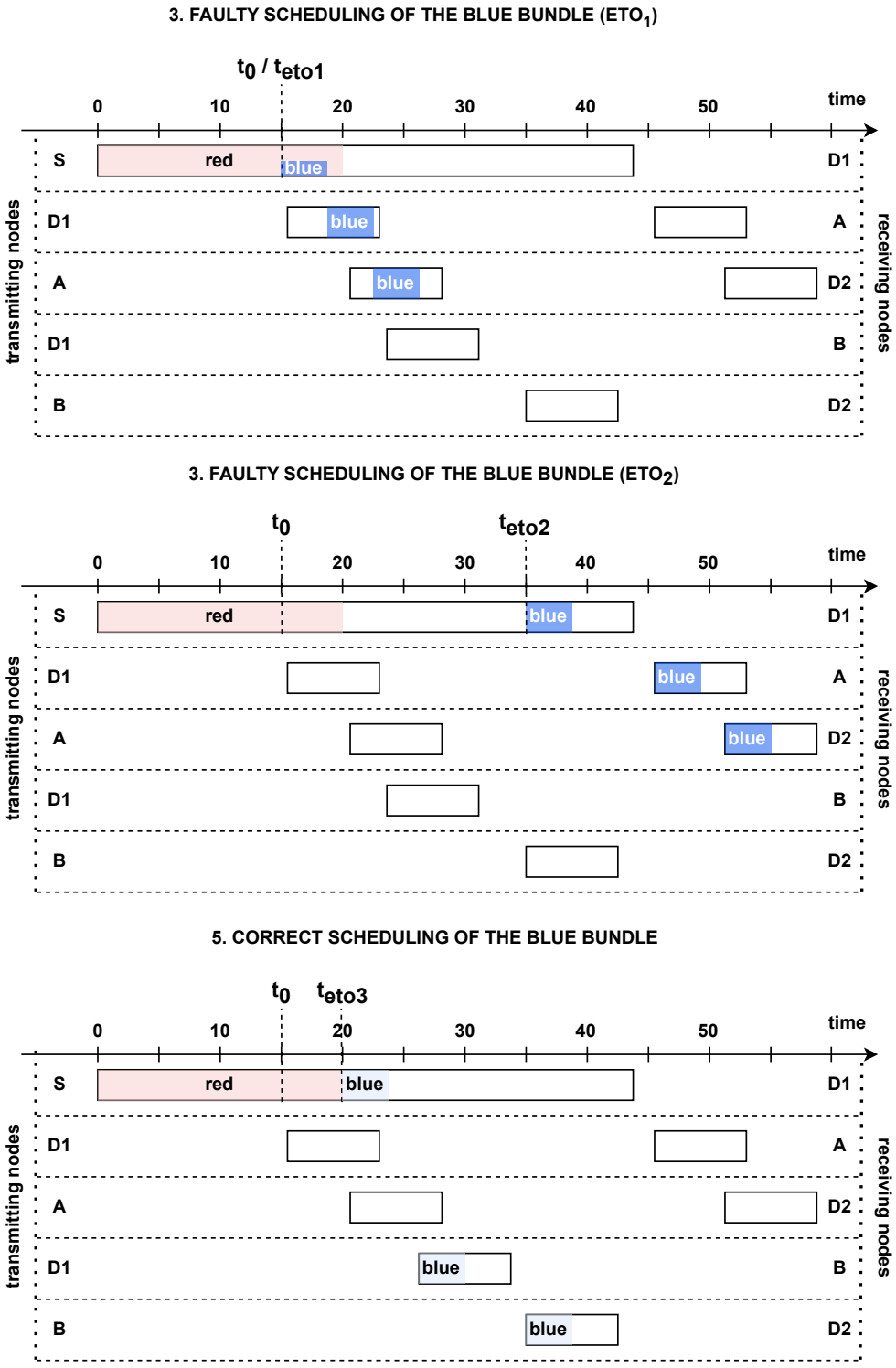


Figure 4.11: Ghost queue issue scenario. The dark blue boxes depict faulty volume booking. Scenario topology is depicted in figure 4.10.

- The red bundle transmission end time is calculated with other means than the current queue analysis (bottom graph). The ETO is then t_{eto3}.

Of course, only the ETO equal to t_{eto3} can be considered correct, as it coincides with the transmission end time of the red bundle and the real earliest transmission opportunity for the blue one.

The scenario allows the bundles to reach their destination, whatever ETO calculation is processed. But any volume awareness inaccuracy can have repercussions on the following bundle scheduling. Correction of a volume booking inaccuracy is only achievable if the bundle has to be re-scheduled locally.

4.6.4 Data rate variations

A physical contact does not necessarily exhibit a constant data rate over time, as mentioned in section 3.1.5 with the characteristic bell shape of the curve of the data rate over time for ground-to-orbiting satellite contacts.

The current solution is to split a contact into several contiguous contacts of different data rates [FDJB21]. The previous sections assumed constant data rates. In this section, splitting a contact means having two distinct contacts in the contact plan.

A bell shape data rate over time would be represented with contiguous contacts showing increasing and decreasing data rates. This approach presents a good tradeoff between simplicity and accuracy.

However, splitting contacts can have an impact on routing for large bundles that cannot be fragmented. In this case, the routing algorithm and the volume management solution shall detect that transmission of a single bundle overlaps contiguous contacts and render the scheduling possible as those contacts represent a single physical link. Such support does not exist at the time of writing.

4.7 Hierarchical interregional routing

The hierarchical InterRegional Routing (IRR) based on [Ale19], modified and implemented in ION [Bur07], addresses internetworking by using specific nodes as passageways between an outer region and a home region. The regions are part of a hierarchical tree structure. IRR passageways are unique for two neighboring regions and shall operate with both contact plans. Organizational concerns shall also be considered, as different actors might operate the regions, while a single actor might operate the passageway. A regional structure example was already depicted in figure 3.2.

The main criticism was that passageways were single points of failure, endangering the interregional routing for the whole network in case of a single regional interface breakdown. If a regional interface failure occurs, all the outer region nodes recursively cannot communicate with any of the inner region nodes. This was addressed by the introduction of passageway failover[6].

Failover does not ensure minimal interregional delays, as it depends on the contacts between the primary passageway and not all available contacts between the passageways and the neighboring region.

The current forwarding table structure (see ION code) and the probing mechanism seem incompatible with multiple passageways. Those aspects of the approach were conceptualized for single passageways at each recursion iteration (at each interface along a path).

[6] Information provided by ION's author.

Last but not least, the separation of concern is suboptimal. The passageways are members of two regions simultaneously and need the contact plans of those regions. In the context of interagency collaboration, this can bring operational tension if a single agency administrates a passageway between two regions operated by different agencies. The agency which does not have administration rights on the passageway would have to supply its contact plan to the administrating agency.

Tension is understandable for an agency that does not operate the strategic node. Also, the administrator node would need to assume the computational cost of running the CGR with a second contact plan for a region that is not his own. Depending on the agencies' relations and their region sizes, such simultaneous membership can be arguably sustainable for administration and operational reasons.

4.8 Other potential issues

Other concerns that were not covered in the previous sections exist:

1. Scheduling of fragments: the routing of bundle fragments is questionable. The Route Volume Limit (RVL) of a route is defined as the smallest value of EVL among all contacts included in the route. That being said, this RVL depicts the maximum load that can reach the destination from the source in a streaming fashion. However, an intermediary node can only forward a bundle if the bundle (or bundle fragment) has entirely been received from the previous transmitter (reactive fragmentation is possible). As a result, the RVL is, in practice, smaller than stated in the SABR standard in the presence of subsequent contact overlapping along the path. The RVL shall be considered as an upper bound, but the need to calculate a Maximum Bundle Size Limit (MBSL) could be appreciable to allow the anticipatory fragmentation.

   If the bundle is fragmentable but anticipatory fragmentation is not appropriate, SABR states that "*again the bundle shall simply be enqueued for transmission to the entry node of the best candidate route*" (SABR, section 3.2.8.1.4, page 3-9), even if the RVL is lesser than the bundle size. This is questionable as there is no guarantee that an alternative route for the fragment can be found on the downstream node that will process fragmentation. In the same way, if anticipatory fragmentation is allowed, a fragment having a size equal to the best route RVL will be created and enqueued for this route without any dry run check. Creating a fragment of a size equal to this hypothetical MBSL would be highly preferable.

2. Calculation of the effective stop time: SABR defines the effective stop time: "*The effective stop time of one of the contacts in a route is whichever is less, the stop time of that contact or the smallest value of stop time among all successor contacts in the route.*" (SABR, section 3.2.6.8.6, page 3-7). This can be understood as the latest sending time from the source at which delivery of a single bit of information is not ensured anymore. But the calculation proposed by SABR is incorrect, as it doesn't take into account possible cumulative OWLT. A trivial example scenario would be a three nodes network (*A*, *B*, and *C*), with A being the source and *C* the destination. Two contacts with start time and end time exist, from *A* to *B* and from *B* to *C*. The SABR stop time would then be the end time shared by those two contacts. However, if the contact from *A* to *B* shows an OWLT, the

expiration of the route shall be earlier to be able to reach node *B* before the stop time of the contact from *B* to *C*.

3. OWLT margin: the calculation of the OWLT margin is used to offset the arrival due to nodes' mobility speed during transmission. This margin is mission-based and appears to be calculated thanks to a global constant in ION. SABR also suggests that this margin is mission-specific. On the contrary, it could be suggested that the margin should be contact-based with correct information supplied via the contact plan, allowing the local node to calculate a fine-grained margin for communications with nodes showing different speeds and ranges.

4. Handling of expiration time: a route can be discarded during selection if the projected arrival time is later than the bundle expiration. A node might have no routes available suitable for such a bundle, even if the routes were not leveraged before for scheduling (i.e., unsuitability is not due to the past scheduling). This is a serious security issue when using Yen's algorithm with an adaptive implementation. It can be imagined that if the expiration time is earlier than the current time, the bundle would be discarded. However, if expiration is later than the current time, it should also be required to force scheduling termination if the last found route by Yen's algorithm had an arrival time later than the current bundle arrival time. SABR does not give precisions about this use case (this might be considered an implementation detail).

   This expiration handling is a concerning issue, as Yen's algorithm can experience a more dramatic hang than the hangs described earlier. If no termination policies are available, CGR keeps searching routes until all existing routes are found, whatever arrival time they have. If the network is large enough (surpassing a couple of hundreds of contacts), and if the attacker has some knowledge about the contact plan, he could paralyze the whole network by sending bundles with early expiration times to trigger a hang on each node in the network.

Those gaps and inaccuracy can justify motivations to be cautious with SABR. The concept developed in this thesis does not strictly follow the standard but respects the best route definition.

The current pathfinding designs, not constrained by SABR (but still influenced), are also problematic, with Yen's algorithm being unusable in large networks and alternative techniques sacrificing networking performance by rendering routes undetectable for computational pressure reduction.

Alongside path detection and selection, the volume management techniques of CGR might also be improved. Various features are deployed to address the same issues at different locations (e.g., ETO for the first hop, queue-delay for the next ones). At the same time, developing a single general and more accurate approach seems achievable and required.

5 State-of-the-art and related work

5.1 Taxonomy

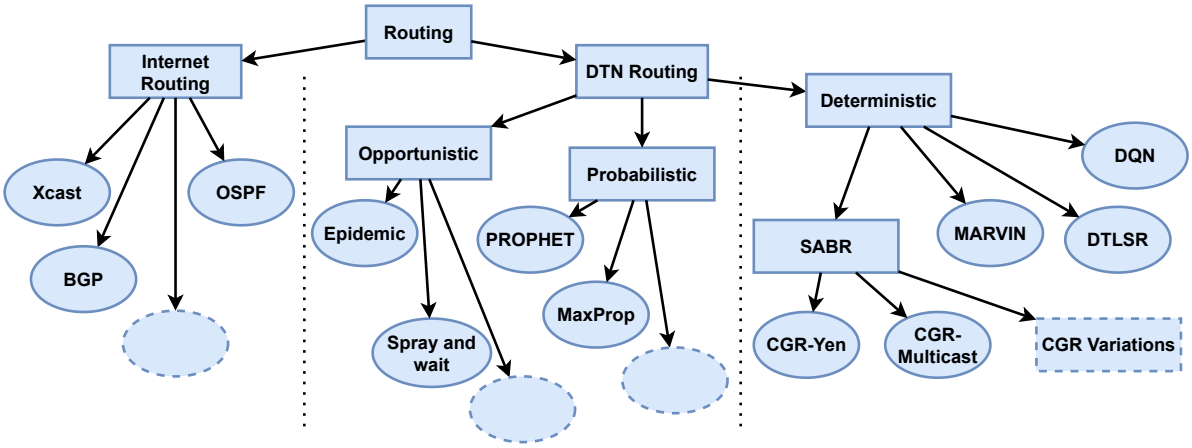


Figure 5.1: Considered taxonomy for the state-of-the-art analysis.

In the context of this thesis, a relatively simplified taxonomy of routing mechanisms will be discussed as depicted in figure 5.1, to avoid more complex classification being too detailed for classes of approaches that are less relevant for this thesis [Mal93, MM97]. The targeted use cases highlight their scheduled nature, which can be highly beneficial for routing purposes.

Deterministic approaches would be preferred, as highlighted by the SABR standard from the CCSDS describing the recommended routing method for scheduled space DTNs. However, opportunistic and deterministic approaches can be used in scheduled DTNs (they would ignore the contact plans); therefore, some of them will be presented in this section. As those approaches rely on replication to increase the delivery rate, congestion risks also increase. In a space DTN, links can show extreme constraints and are precious resources; replication-based algorithms are consequently not the first choices.

Even though this thesis focuses on schedule DTN and the application of efficient deterministic behavior, an unplanned event like node failures might bring the need to fall back to a non-deterministic routing strategy (this remains speculative, e.g., if the contact plans cannot

be supplied or corrected on time). The motivation to present the following approaches is twofold. On the one hand, mentioning the available backup options is essential if deterministic routing is no longer possible. On the other hand, to describe more precisely why those approaches are sub-optimal on scheduled DTNs, even though they are applicable.

5.2 Opportunistic and probabilistic approaches

5.2.1 Flooding approaches

The most trivial approach is to flood the network by sending messages via each available contact, eventually reaching a point when one copy reaches the destination. This approach was initially coined epidemic [VB+00], and the core principle can be described with various nuances.

For instance, spray-and-wait [SPR05] is an epidemic-like approach with the difference that the number of copies cannot exceed a given value. Again this sub-approach itself shows variations, limiting, for example, the number of hops by creating half as many copies as at the previous hop, given the last hop replication count.

The epidemic approach has also been adapted to multicast purposes in [JWZS10] with the Epidemic-based Controlled flooding and Adaptive Multicast for Delay Tolerant Networks (ECAM).

The flooding approaches are known to be very simple and show very little computational pressure. However, the replication pressure can induce congestion. Another aspect is the exchange of summary vectors at the start of a contact to notify the other node of the bundle stored locally. It permits avoidance of any transfer of bundle already present in the receiver's storage. This assumes that all contacts are bidirectional, which is not always the case, and this limitation is shared with all vector-sharing-based approaches.

5.2.2 PROPHET

In a DTN, the node mobility is not necessarily unpredictable, and mobility or contact patterns can emerge. If the mobility patterns are entirely random, flooding the network is a legitimate approach, as the subsequent contacts with a given node are hardly predictable. The Probabilistic ROuting Protocol using History of Encounters and Transitivity (PROPHET) [LDDG12] leverages the non-random nature of the mobility patterns to reduce the replication pressure.

Each node locally maintains a Delivery Predictability (DP) value per neighbor, updated during each contact. If the local node frequently encounters another neighbor node, the DP value for that neighbor will increase. In the same way, if no contact is observed with another neighbor, the DP value for this neighbor will decrease.

Moreover, the nodes exchange all their local DP values when a contact occurs. Receiving the DP values from the neighbor allows the approach to recalculate the local DP values transitively.

The forwarding process is then simple: the node will only transmit a message to a neighbor having a higher DP value for the message's destination. This approach shows a lesser congestion risk but is not resilient to mobility pattern inconsistency.

5.2.3 MaxProp

MaxProp presented in [BGJ+06] is very similar to PROPHET. It, however, introduces specific queue management. Its main function is to sort the bundles by destination, ordering the destination by their Delivery Likelihood (DL).

Another main difference with PROPHET is the interdependence of the DL probability values. The sum of those values shall always be equal to one. The local node stores then in a vector a delivery likelihood for each other node in the network. When a node meets another node, they exchange their vectors, and each node stores the vector of its neighbor.

And last, MaxProp leverages Dijkstra's algorithm, proceeding to actual probabilistic end-to-end pathfinding using the vectors. As Dijkstra's algorithm finds the lowest cost path, the delivery unlikelihood values are instead calculated, and the route cost is the sum of these values along the path.

MaxProp also provides other priority rules. For instance, the messages destined to the neighbor are transferred first. Also, messages that have not traversed far in the network are given priority. This provides the benefit of creating copies of a message as early as possible to increase its chances to be delivered instead of having it lie dormant, for an arbitrarily long period, in the storage due to long forwarding queues.

5.2.4 Issues

Regarding the contact volume representation, the requirements /R4/, /R5/, and /R6/ are not likely to be fulfilled, as the local node has no information about the connection termination with the neighboring nodes, and thus cannot calculate available volume.

In the same way, as long as no end-to-end paths are calculated at each intermediary node, the requirement /R1/ is also non-applicable.

As long as some protocols like the flooding-based approaches can be used for multicast and unicast bundles, the requirement /R3/ can, in some cases, also be considered fulfilled.

The opportunistic approaches usually show low computational pressure for the next-hop selection, fulfilling the requirement /R2/.

Given the large majority of the requirements that are not fulfilled, including all the CCSDS recommendations about routing within a scheduled DTN, the opportunistic approaches will all be discarded for the application domain of this thesis (SABR-based algorithms for the optimized delay and reduced congestion risks, targetting scheduled DTNs).

5.3 Deterministic approaches

5.3.1 Movement-aware routing over interplanetary networks

The Movement-Aware Routing oVer Interplanetary Networks (MARVIN) approach, presented in [SMM04] also relies on Dijkstra's algorithm implementation for pathfinding. The contact plan creation and the routing algorithm were not considered two different concerns. In contrast, modern approaches assume that the contact plans are just provided to the nodes with external mechanisms.

This approach is relatively primitive, as MARVIN updates a single routing table in a classic Internet fashion, with a single entry for each source and destination pair. To support the link breaks, MARVIN includes proactive route reconfiguration concepts.

This approach could work correctly with small packets but not with modern DTN bundles. Even if somewhat ambiguous, the link breaks shall also be dependent on the bundle sizes. More precisely, the MARVIN approach maintains a single route for a destination, regardless of the actual bundle to send, and if the bundle is big enough, what could be related to a link break, shall occur earlier to recalculate a route for this bundle.

On a rather serious aspect, the original paper does not provide any simulation results, only the routing table calculated by MARVIN. Also, MARVIN calculates the subsequent contacts between the planets with orbital trajectory analysis and so works with planet nodes. This is insufficient to derive fine-grained contact properties depending on the hardware capabilities, for instance, to calculate the volume allocation from the link data rates and the bundle sizes.

5.3.2 Delay-tolerant link state routing

Delay Tolerant Link State Routing (DTLSR) presented in [DF07] attempts to adapt, for the DTNs, Internet link state protocols such as OSPF [FLM08]. The routing of a message is processed with a delay-tolerant Dijkstra's algorithm for pathfinding, and so exhibits the same base functioning as SABR. The presence of regions called areas (as in OSPF) was already introduced to reduce the pathfinding processing time.

Another reason for separating the network in areas is the absence of some central oracle entity to distribute the contact plans to the network nodes. The nodes themselves would flood link state announcements, as smaller subnetworks would probably increase the performance of such announcements.

This approach shows two main issues. First, the DTN constraints do not support the scalability of the delays within a single area. The link announcement propagation delays would result in inaccurate contact awareness for distant nodes. Also, the paper does not support time-varying graphs, i.e., the network graph is just a snapshot of the current link state knowledge for a given local node. For the unavailable links, the outage is capped at 24 hours. This is a questionable choice for scheduled DTN, as the next connectivity opportunity shall be known in advance, thanks to the contact plan. This makes DTLSR more suitable for MANETs, and less adapted for scheduled space networks.

5.3.3 DTN routing for quasi-deterministic networks

The DTN routing for Quasi-deterministic Networks (DQN) approach, presented in [DLF+17], leverages the nodes' positions for routing within satellite constellations.

The interesting aspect is the different usage of node mobility knowledge. Instead of deriving contact plans from the node mobility predictions and processing the shortest-path algorithm against time-varying graphs, DQN uses directly the node positions.

The source sends copies fulfilling an alternation, marking each copy with its physical transfer direction, east or west. The intermediary nodes search the physically closest neighbor K to the destination, taking the direction of the bundle into consideration. The intermediary nodes can also request the contacts of their neighbors to compose a larger set to search the closest neighbor K (so K can be at a two hops distance from the intermediary node). DQN then forwards the message directly to K or to the neighbor showing a contact with K.

5.3.4 Issues

The MARVIN and DTLSR approaches benefited the field as pioneers in developing deterministic approaches for scheduled DTNs. They are, however obsolete, in the case of MARVIN, because of its lack of practical use case and its lack of dynamic behavior that does not address clearly the problem, or inapplicable in the case of DTLSR due to the higher delays of space networks that cannot provide the required reactiveness when using state announcement techniques.

The problem with DQN is that the algorithm has a two-hops depth knowledge, while the average-path length can show high variation and range from a few hops to more than 20 [FW17]. This study, however, used CGR, which minimizes the end-to-end delays.

Those approaches are not suitable to fulfill at least partially the requirement /R4/, which is unacceptable for deterministic approaches addressing the application domain of this thesis.

5.4 CGR variants and enhancements

5.4.1 CGR alternative routing table computation

As already stated in a previous section, the first-ending and first-depleted approaches have been presented in [FMCF18] as alternatives to Yen's algorithm, but again those methods are likely to suppress contacts which could be relevant for the following Dijkstra's algorithm calls, and therefore those methods are likely to miss some very interesting routes, and in the worst case, the best existing one included.

The same paper also presented two other approaches, namely the one-route and per-neighbor-route approaches.

For the one-route approach, the paper states that "*the one-route approach drastically increases the computation efficiency as the outcome of every Dijkstra calculation is effectively used to assist in the forwarding of bundles*".

This statement is likely true in comparison to Yen's algorithm, the first-ending, and the first-depleted approaches if the number of nodes or contacts explodes. However, this remains heavily dependent on the load's nature, but as long as large bundles might be preferable, this statement can hold for a wider scope. This statement can also hold if the computational time of a single route remains negligible. In this case, "negligible" can hardly be described with numbers, as this probably depends on the node computational power and the operator expectations. But using less fine-grained criteria, one can assume that a routing decision processing time within the millisecond range is acceptable, while one requiring several seconds to minutes would be unacceptable. However, the paper does not mention any special considerations about Dijkstra's algorithm implementation but seems to use a design similar to the one from ION-CGR, which can, in some cases, output a route that is not the best existing route.

The per-neighbor-route approach acts as an intermediate between the static approaches, which calculates the possible routes beforehand to populate the routing table, and the on-demand approach, like the one-route approach. When operating with the per-neighbor-route design, the local node stores for each destination at most *N* route in the table, *N* being the number of neighboring nodes, each route being computed with the neighboring node being the first hop receiving node (by suppressing the other first-hop contacts). This intends

to support critical bundles, which are '*expected to be forwarded through all possible paths to a destination*'.

Concerning volume management, the same paper introduced the volume updates of all the contacts along a path, which is now part of the SABR standard. But as already stated in the previous sections, it is only with the queue-delay enhancement, presented in [CDCP21], that the volume updates are taken into account to calculate more accurate projected delivery times for the next route selections. However, this approach does not cover all edge cases.

5.4.2 CGR-multicast

CGR-multicast [DJ19, Jon19] is a feature proposed in 2019 and then incorporated in ION-CGR for the release 4.0.1, and is also an adaptation of eXplicit Multicast (Xcast) [OFI+07]. This approach, built on top of ION-CGR, supports all of the current modern DTN mechanisms. The bundle protocol does not provide any multicast header capabilities. It has then been decided to use an extension block to carry the destination responsibility list for a given bundle.

This approach is inspired by Dynamic Tree-Based Routing presented in [ZAZ05]. Even though old, this approach gathers the current ION-CGR multicast base principles for non-replication multicast routing in space-scheduled DTNs. A multicast routing algorithm follows the non-replication paradigm only if duplication is meant to support downstream path branching and not processed in other cases. DTBR leverages a backend unicast routing approach using the Dijkstra's algorithm from [JFP04] and uses on top a header mechanism similar to Xcast.

CGR-multicast is independent of the underlying unicast pathfinding approach, i.e., the operating CGR variant for unicast.

In CGR-multicast, the destination responsibility list is referred to as a gang, and the process of creating a gang for a downstream branch is called ganging.

This approach has been adopted to replace the historical static multicast tree of ION, but it still shows some weaknesses. Firstly the ganging process uses the unicast route selection process to find the best routes, forwarding the bundle to the first hop of the best route to each multicast destination. The ganging process then merges the first hop and destination pairs into a list of first hops with a list of destinations (a gang) attached to each of these first hops. The concept is not necessarily problematic; however, the route selection process is expected to show a higher cost in large networks ($\mathcal{O}(N^2)$ instead of $\mathcal{O}(N)$).

Secondly, volume updates are not supported in a fully SABR-compliant manner. In 2019, this was not necessarily considered a problem, as ION-CGR was still updating only the first hop contact upon scheduling a unicast bundle. But now SABR specifies that the volume of the contacts along the bundle path shall be updated.

It is interesting to point out some slight variations observed in some very related approaches regarding the management of the header (extension block in the case of CGR). In Differential Destination Multicast (DDM) [JC01] or On-demand situation-aware multicasting (OS-multicast) [YCCD06], the header management is also meant to support opportunistic behavior.

DDM, for instance, only describes quite sophisticated header management with soft-state capabilities. In the soft-state mode, the nodes retain some memory about the tree and become responsible for the delivery to a certain set of destinations. In soft-state mode, the DDM header also acts as an administration message to notify downstream nodes of the new set of destinations they should take care of. This approach is not incompatible with

the space network high delays as long as a given destination responsibility list of a receiver remains relevant for a single transmitter.

In OS-multicast, the header carries a tree rather than a destination list, but the intent can be considered equivalent. The main difference is that OS-multicast allows bundle transmission to nodes that are not on the depicted downstream path. This allows to leverage of opportunistic links to increase the delivery ratio and decrease delay, but the approach cannot be considered a non-replication multicast approach.

Space Minimizing Tree-Based Routing (SMTBR) presented in [Tri13] is related to CGR-multicast but intends to reduce congestion. The backend shortest path algorithm computes thin trees, reducing the number of participating nodes and de facto reducing the number of bundle copies needed to reach each destination.

The tree can be constructed with two different variants, both leveraging the election of important nodes, which are more likely to be included in a path. This algorithm shall also ensure that the nodes agree on the paths for accurate volume management and reduce the showing risk of network loops (leaving aside the unforeseen possible downstream congestion). This seems achievable as long as the coefficients used to calculate the node importance property are identical on each network node.

Again this approach brings questioning regarding its possible implementation for modern DTN. The bundle protocol does not present any multicast header (but an extension block can achieve the same goal as seen with CGR-multicast). The tree construction does not conform to the SABR standard expectations. Certainly, creating a thin tree requires some trade-offs regarding the bundle delivery delays, while SABR requires that the fastest route be selected in the routing table.

5.4.3 CGR extensions

CGR-EB [Bir11] is an extension of CGR that proposes to attach the whole path of a bundle in an Extension Block (EB) from the source. With such a mechanism, the processing time is spared for the next hops, as the forwarding nodes can just check the extension block for the next forwarding decision. More than a processing improvement, a source node might have a better view of the topology than the next nodes along the path, justifying a pathfinding decision from the source leveraged by the forwarding nodes.

CGR-BF [DBDG18] and CGR-SPI [DGDG19] are two extensions that intend to optimize volume management. CGR-BF proposes to optimize bundle fragmentation strategy, while CGR-SPI proposes queue occupancy sharing with the neighboring nodes.

A common characteristic of those extensions is their apparent compatibility with any underlying pathfinding algorithm for CGR-EB and any underlying volume management caring algorithm for CGR-BF and CGR-SPI.

Opportunistic capabilities were also proposed in [BBC+17], to allow CGR to leverage discovered links. This capability is considered out of scope as long as this thesis focuses on predicted contacts.

5.4.4 RUCoP and CGR-hop

The Routing under Uncertain Contact Plans (RUCoP) approach depicted in [RFM+21] leverages the Markov decision process over a Markov chain graph depicting the probabilities of the contacts to extend CGR for operation under inaccurate contact plans.

CGR-hop mentioned in [RFM+21] is also a pathfinding modification (and not an alternative path construction method), which proposes to modify the metrics priority by lowering the hop count first during construction. Consequently, contact utilization is reduced, but delays are increased.

RUCoP targets networks with inaccurate contact plans, while CGR-hop targets congested topologies. In this thesis, the contact plan as accurate and inaccuracy countermeasures are left for future work (or by introducing existing extensions to the proposed approach). CGR-hop does not respect SABR's expectations.

5.4.5 Issues

Approach	/R1/	/R2/	/R3/
CGR-yen	◑	○	◑
CGR-limiting	◑	◑	◑
CGR-one-route	○	●	◑

Table 5.1: Requirements fulfillment relative to volume pathfinding by the state-of-the-art approaches.

Approach	/R4/	/R5/	/R6/
ETO	◑	◑	○
Queue-delay	◑	◑	○

Table 5.2: Requirements fulfillment relative to volume management by the state-of-the-art approaches.

Regarding the networking performance, /R1/ could only be fulfilled by CGR-yen. The fulfillment of the requirements by CGR can be found in tables 5.1 and 5.2. The symbols ●, ◑, and ○ depict sufficient support, partial support, and no support, respectively. However, adaptive behavior can lower the selection quality. The limiting contact approach (CGR-limiting in the table) can also show non-negligible side effects by suppressing important contacts. Retaining the routes can permit, in some cases, to achieve better selection (by providing candidates). In opposition, the one Dijkstra per bundle approach (CGR-one-route) would suffer from inaccurate pathfinding at each bundle scheduling if the current element is a node, or increased processing time if the current element is a contact. The /R2/ is not fulfilled with CGR-yen. CGR-limiting on its side shows a fewer computational and memory pressure. The one-route approach has an inexistent memory pressure, but the computational pressure can be high if the bundles are small and plentiful. Multicast is an extension of CGR and inherits the behavior of the underlying CGR implementation. The one-route approach would need to compute for each bundle one route per multicast member, which can, in some cases, be considered worse but not comparable to the risks of hangs with CGR-yen. In all cases, volume management is an issue if using CGR for multicast due to single destination pathfinding.

The volume management, as performed with the EVLs for different priorities, ETO, and queue delay, was shown to be inaccurate in some special cases but effective in addressing volume management in other cases.

The optional CGR-EB, CGR-BF, and CGR-SPI features do not affect the pathfinding of a single bundle fragment. In the case of CGR-EB, the path is just attached to the bundle. In

the case of CGR-BF, a different fragmentation strategy is applied before the scheduling of the fragments. In the case of CGR-SPI, a new means for volume management updates is proposed. As those extensions use the same pathfinding algorithm, they are not likely to increase the scalability of CGR as they do not modify its core characteristics (processing time, pathfinding, and volume management).

RUCoP operates in a different use case and cannot be considered here, while CGR-hop modifies the end-to-end transmission expectations. CGR-hop is compatible with all CGR variants depicted in tables 5.1 and 5.2. It would exhibit the characteristics of leveraged variant for the considered requirements and therefore do not motivate the addition of a dedicated row in the tables.

5.5 Interregional routing

Even with a drastic reduction of the computational pressure of deterministic routing, it is not expected to be able to scale the network indefinitely without the need to introduce subnetworks. A space DTN subnetwork is called a region, and interregional routing is expected to be a key component for scalability. The following section covers approaches that are related to interregional routing.

5.5.1 Border gateway protocol

The Border Gateway Protocol (BGP) [RHL06] is an Exterior Gateway Protocol (EGP) and key Internet technology. However, this protocol is not expected to perform correctly in a DTN. The protocol usually relies on sessions, which is hardly compatible with delays. Acknowledgment on session opening, closing, or failure might not be received in time to actually allow those sessions to be leveraged.

For example, if we assume the regional structure to be topology-based, the session between two BGP routers (e.g., one on Earth and one on Mars) orbiting their planets may hardly get the chance to make this protocol work appropriately as the session management time are not negligible when compared with the potential contact durations (Earth LEO satellites orbit periods are about 90 mins, Earth-Mars delays of about 20 mins on average). BGP organizes the exchange of routing information but will hardly provide efficient end-to-end routing capabilities from a source to a destination at a given time when operating in a DTN.

5.5.2 Hierarchical interregional routing

Interregional Routing (IRR) was already covered in sections 3.1.6 and 4.7.

Hierarchical Interregional Routing was proposed in [**?**] to provide EGP capabilities for hierarchical partitioned space networks. A gateway (called a passageway) is a bridge node between two regions. It is part of the two regions simultaneously, the home region, which is accessible from outer regions only via this passageway. A home region can encompass several inner regions, accessible via nodes being passageways for those inner regions. This home region is consequently the outer region of the two regions those passageways are part of, the inner regions being their home regions.

Routing is quite simple. If the source knows the destination, i.e., the node is part of the contact plan, CGR is used. Otherwise, nodes access forwarding tables that provide, for an

interregional destination, the next passageway along the end-to-end path. If such an entry is absent, probing is triggered.

5.5.3 Issues

On the one hand, BGP should be disqualified entirely right away due to its absence of delay-tolerant features. On the other hand, the use of a Multi-Exit Discriminator in BGP seems plausible, and an analog approach would be highly beneficial for DTNs, to address dynamically the requirement /R7/.

This requirement can only be fulfilled partially with the current IRR concept, as single nodes act as passageways (no multiple passageways simultaneously). The single point of failure issue was addressed in ION with failover support. A passageway that proved to be the best option during probing (by receiving the acknowledgment first by one of the passageways) will be selected for IRR forwarding for the next bundle sharing the same destination. As soon as forwarding to this passageway fails, failover to another passageway is organized. The code base for the IRR support is still under development in ION, and no publications are available to support this section. Thus, this information shall be taken with care.

However, load balancing-like behavior, to increase throughput and decrease interregional routing delays, shall allow sending an interregional bundle to the best passageway of a neighboring node for a given current time. No evidence of such support was observed at the time of writing.

IRR uses a relatively static tree, rendering it impossible to leverage shortcuts. The requirement is /R8/ unreachable for fulfillment. The introduction of shortcuts could be organized if a network loop control is provided. Assigning specific nodes (the passageways) for interregional forwarding does not seem flexible. If failures occur, and if the passageway node becomes a node that was not a passageway before, previous probing phases are rendered useless as the IRR forwarding tables point to nodes that are not operating anymore.

The simultaneous presence of two region contact plans on a single node also conflicts with requirement /R9/.

5.6 Further approaches

Those approaches cannot be considered ready for comparison with CGR, due to a lack of performance or the ongoing development they are subject to. They are, however, presented here for convenience.

5.6.1 Machine learning approaches

The increasing interest in the machine learning field also overflowed to the DTN routing domain. However, machine learning for schedule DTNs can still be considered to be in an early stage, with critical issues not addressed to this day.

In [DHP17], the evaluation does not include CGR, and the reinforcement learning approach is only evaluated for the end-to-end delay against a shortest-path algorithm without clear SABR compliance statements. Also, the proposed algorithm seems to only replace the route selection process, while most of the issues related to SABR come from pathfinding. The real CGR issues detected in this thesis are computational pressure, the best route detection, and volume management. As those issues are closely entangled with route construction,

replacing only the route selection might be insufficient. Another reinforcement proposal in [SOBH16] proposed a more extensive networking evaluation, but CGR is not evaluated at all.

In [DP18], a supervised learning technique is leveraged, but only the machine model is evaluated, and no networking evaluation is provided, in comparison to [Len20] where a spiking neural network is leveraged and evaluated against CGR. However, the topology used for evaluation is questionable. The absence of congestion or dead-ends allows the bundle to eventually reach its destination, whatever forwarding decision is applied. The response times are also very low in comparison with the other approaches, and an interpretation of this behavior is complicated, as SABR is supposed to optimize the end-to-end path. As the training method is unclear, the reasons why this approach can surpass CGR are also unclear.

The main issues shared among those approaches are mainly the evaluation strategy and information about the dependence on training and responsiveness to unplanned node introduction and the networking capabilities of those approaches just after a contact plan update. The training is also crucial, as training against CGR routing decisions might be sub-optimal regarding CGR next-hop selection issues.

5.6.2 Tropical geometry

Tropical geometry is a mathematical field that has already shown interest in optimization problems, e.g., in the domain of transportation networks [Kri14].

Geometric solutions were historically intuitive in the train scheduling field, as the trains could be represented as points in a space-time plan (their lengths being omitted). Drawing a line translates the movement of the train when having a constant speed from one station to another (also being lines perpendicular to the time axis), and scheduling can be processed by drawing the train lines. If two lines cross, there is a collision, i.e., the scheduling is faulty. Such geometric solutions were proposed as early as 1878 [Mar78].

Direct mapping of DTN scheduling with such a geometric solution would be more complicated, as the rails (contacts) would not be static, and the trains (bundles) would not have a negligible length (bundles might be large).

Recently introduced by [CHS+22, SHC+22], tropical geometry solutions are being developed for DTN scheduling but can be considered in an early stage still. No routing algorithms are available today for networking evaluation as those publications propose, for the moment, theoretical advancements in the field rather than practical advancements.

6 Scalable schedule-aware bundle routing

6.1 Overview

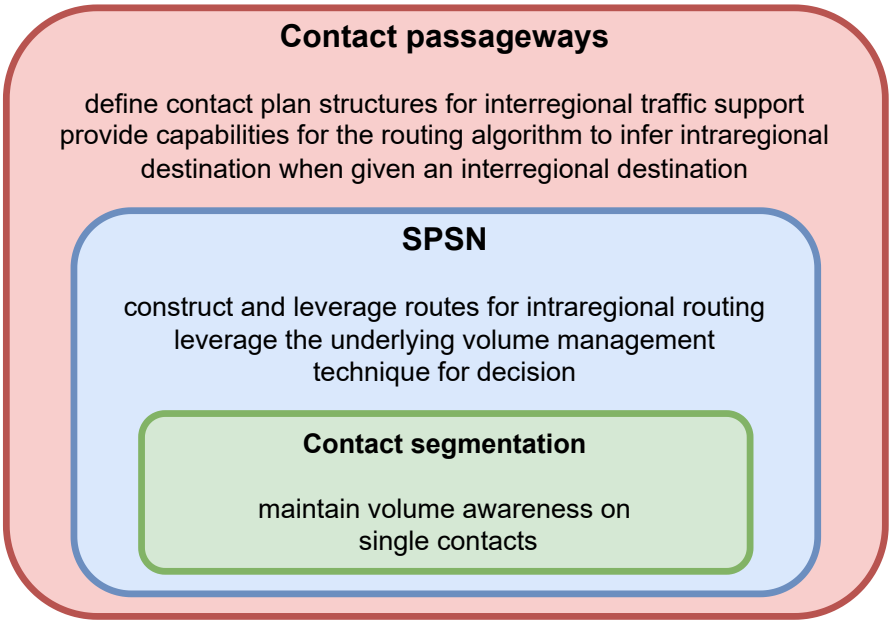


Figure 6.1: Relations between SPSN, contact segmentation, and contact passageways.

Three relatively independent approaches are proposed to provide a scalable solution for the addressed domain of space networks. Although they can be deployed separately to replace only sub-parts of the state-of-the-art approach, they can be joined together to form a routing approach addressing the three main topics of this thesis (as depicted in figure 6.1):

- Shortest-Path tree routing for Space Networks (SPSN) for intraregional routing scalability.
- Contact segmentation for volume management scalability.

- Contact passageways for interregional routing scalability.

Shortest-Path tree routing for Space Networks (SPSN)

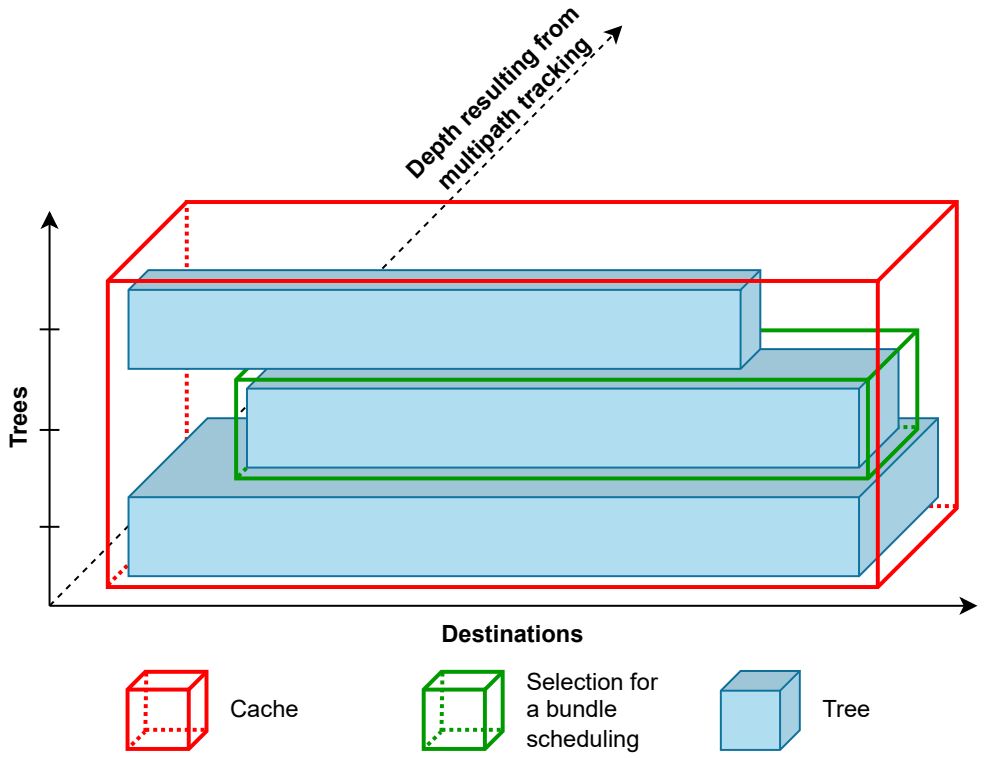


Figure 6.2: Representation of the route container within SPSN.

SPSN proposes a paradigm shift to organize the routes container by layers, as shown in figure 6.2, in comparison to the single destination-based memory blocks of SABR (figure 3.3). SPSN merges the routes alongside the destinations axis by computing shortest-path trees instead of single destination routes. Consequently, all routes of the same tree (i.e., same layer) share the same characteristics. The layers can show variable length alongside the destinations axis, as some nodes might be unreachable due to the constraints supplied for specific tree construction.

A shortest-path tree presents up to $V-1$ edges for a V nodes graph and is not supposed to overlap three dimensions, as the retainment of the parent for each vertex is sufficient. The need for this third dimension is part of the multipath tracking feature, covered in section 6.2.2.

The cache typically shows an upper bound for the number of trees it can store to limit memory usage. Each tree is stored with the constraints used for its computation, and the tree selection procedure leverages that information for the decision.

Contact segmentation

Contact segmentation proposes a more strict and accurate way to book volume on a contact. This shifts the management from a residual volume and cumulative booked load perspective

to a time interval booking mechanism. Contact segmentation presents the advantages of addressing the bandwidth utilization at any contact, even for the first hop, as the cumulative transmission time required for the bundle enqueued (first contact) or the booked load (next contacts) is now encoded in the same fashion on the contacts.

Contact passageways

The contact passageway design also proposes a relatively significant shift from the existing node passageway design while relying on similar administration management. The shift resides in the dynamic assignment of the role of a passageway to a node for interregional transmission while allowing those nodes to remain members of single regions. Any node acts as a border node if it is a transmitter or a receiver of a passageway contact. Border nodes do not require specific configuration. This naming differentiation is only a matter of convenience: the passageway contacts are not intrinsically special, as the contact plan is still unique for a given region (all nodes use the same contact plan). No limitation can be observed regarding the border nodes. A node can be a border node and sustain interregional traffic intermittently each time it is the transmitter or receiver of the passageway contact. It can also sustain interregional traffic only once (due to the occurrence of a single passageway contact involving this node) before the contact plan horizon.

Moreover, the number of passageway contacts between two regions can be arbitrarily high, with high flexibility in the involved node designation (more than one border node can exist on each side). Node configuration is identical from one to another within the same region.

6.2 Shortest-path tree routing for space networks

6.2.1 Structure

Shortest-Path tree routing for Space Networks (SPSN) intends to reach the scalability goals with high accuracy (reaching state-of-the-art networking performance) and acceptable computational pressure (being free of hangs and being part of the same order of magnitude).

Computational performance highly constrains scalability, and positioning a cursor accordingly between accuracy (e.g., recomputing the best route for each bundle) and reusability (reusing the same cached route as long as possible) is challenging. Further implications are the absence of useless route computation and effort sharing. Observing the unsuitability of a new path after its computation is suboptimal. Negligible overhead computation supporting a wider scope can prevent future computation for other needs (i.e., a tree for several destinations).

For accuracy, SPSN ensures that any computation is relevant for at least one bundle thanks to its volume awareness within its Dijkstra implementation. This discards any useless computations. For effort sharing, SPSN computes shortest-path trees to be used for unicast or multicast bundle scheduling. For reusability, SPSN stores trees that can be elected for reuse depending on the bundle and tree characteristics. In other words, a single Dijkstra call can suffice for multiple bundle scheduling, even if they do not share the same destination.

The following aspects are core to SPSN:

- The tree construction, with a modified version of Dijkstra's shortest-path tree algorithm, allows tracking of simultaneous paths to the same node or optimized contact utilization (multipath tracking).
- The tree management, caring of operations applied to single trees for routing decisions. It processes calls to the underlying volume management functions for dry runs and volume booking.
- The tree caching, caring of the storage policy, and the reuse strategy.

The implementation of an SPSN flavor is therefore adaptable by selecting a tree construction backend (three available), the volume management backend (two available), and the cache management backend (one available with configurable size). For the latter, further flexibility is possible by defining a maximum cache size to fine-tune the memory usage, accuracy, and computational pressure.

SPSN does not leverage routing tables. This key conceptual shift will be covered in the following sections.

6.2.2 Tree construction

Three different flavors are available for tree construction:

- The basic tree construction creates a simple tree following a standard Dijkstra implementation with node-based current element retainment.
- The multipath tracking tree construction creates a tree variant allowing the tracking of several paths to individual nodes simultaneously. The goal is to get the accuracy provided by maintaining a current contact during exploration while operating as if a current node was maintained (please refer to section 4).
- The hop tree construction creates a simple tree prioritizing minimal hop counts.

The constructions of those three flavors are covered in the following sections, with the base tree construction as an introductory flavor describing tree construction basics required to tackle multipath tracking tree construction. The hop tree construction is also provided as it is leveraged as a control during the evaluation.

Basic tree construction

SPSN proposes to create the working areas during the construction, similar to what is suggested in algorithm 1.

The work areas are termed RouteStage in SPSN. A RouteStage is attached to a given node with the mean of a map, providing a reached destination as a key, the best RouteStage detected until this state of the algorithm. A RouteStage is similar to a CGR work area, carrying similar metrics (e.g., arrival time, hop count, and expiration). The parent of a RouteStage is also a RouteStage (and not a contact of a node). This permits the implementation of various flavors flexibly (e.g., maintaining a current contact rather than a current vertex is rendered possible).

The paths could be retrieved from the reverse paths encoded by the RouteStage parenting, but it is not processed at this algorithm stage.

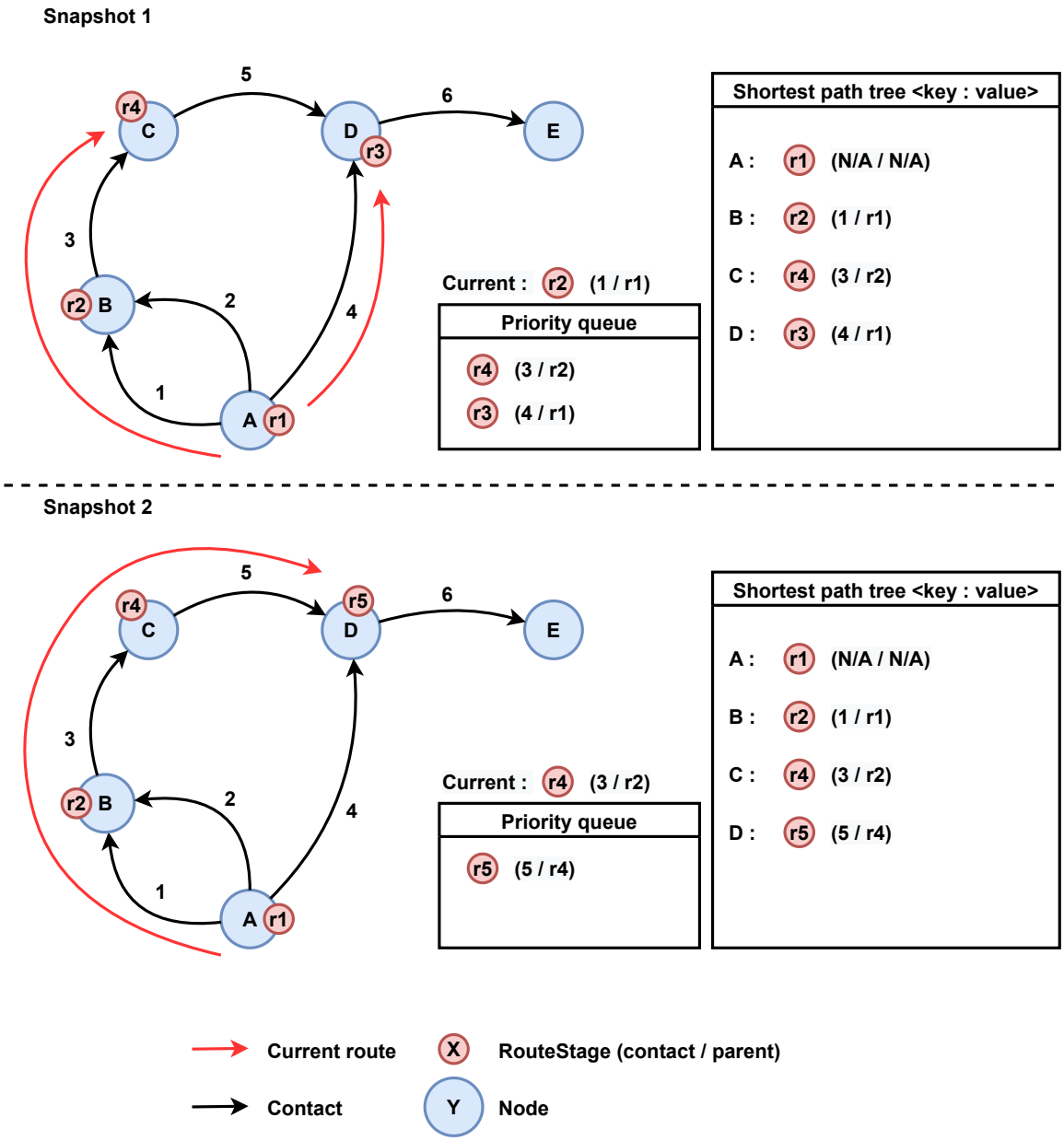


Figure 6.3: State snapshots of the basic tree construction.

Figure 6.3 depicts two snapshots taken during the iterations of the basic tree construction algorithm. The RouteStages are sorted by arrival times: $r1 < r2 < r4 < r5 < r3$. Both snapshots are taken at the end of an iteration (i.e., just before modification of the current RouteStage for the next iteration). The current element is provided, $r2$ in snapshot 1 and $r4$ in snapshot 3. The Priority queue container stores the next RouteStages eligible to be the next current element. The shortest path tree container depicts the abovementioned map.

In snapshot 1, a path to *C* is discovered from the current RouteStage $r2$ being a path to *B*. This newly found route is encoded as RouteStage $r4$ and was pushed into the priority queue. The best path to *C* was updated accordingly in the shortest path tree. This is not depicted in the figure, but there were no routes to *C* before this snapshot, the same way *E* is not part of

the map keys, as *E* still needs to be discovered.

For the next iteration, the shortest distance (earliest arrival) RouteStage was retrieved from the priority queue to be the new current element (*r*4), and snapshot 2 describes the state at the end of this iteration. It can be seen that a new path to *D* was detected and encoded as *r*5. This RouteStage appeared to have a shorter distance to *D* as the former best path to *D* encoded with the RouteStage *r*3 was replaced in the map. When such an event occurs, the replaced RouteStage can also be removed from the priority queue if still part of it, which was the case, as seen in snapshot 1.

Another important aspect is the ability to supply a bundle size for the tree construction to filter paths that would not allow the bundle to reach the destinations. This is referred to capacity oriented search (or volume-aware search) and is a key component of SPSN. This extends the one-route approach to trees.

Because of this bundle size awareness, a single-hop metric calculation is not necessarily trivial, depending on the concepts deployed for volume management. This shall be considered a volume management concern, and it is assumed here that a candidate RouteStage can be created thanks to the volume management framework. For example, creating RouteStage *r*2 retained contact 1 rather than contact 2. This selection is processed on the one hand with the help of appropriate sorting (by start time) of the contacts between *A* and *B*, and on the other hand, with the help of volume management, to discard unsuitable contacts.

In other words, sorting the contacts allows the algorithm to ignore the next contacts as soon as a contact can be selected (a variation of this approach is required if the contacts overlap). While the volume management framework allows proceeding dry runs on individual contacts for exploration.

The first tree construction backend provided by SPSN is this basic pathfinding algorithm, implementing strictly Dijkstra's algorithm.

Multipath tracking tree construction

As an alternation to the basic tree construction, MultiPath Tracking (MPT) intends to address the accuracy issues inherent to current node retainment with a modification to the core of the Dijkstra algorithm.

Destinations	Single destination best path	Shortest-path tree parent	Multipath-tracking tree RouteStage (parent)
A:	S A	S	A(S)
B:	S A B	A	B(A)
C:	S A B C	B	C(B)
D:	S F D	F	D(F1)
E:	S E	S	E(S)
F:	S A B C F	C	F2(C) F1(S)
	(Depth up to 5)	(Flat)	(Depth up to 2)

Table 6.1: Path structures for single destination paths and shortest-path tree.

Multipath tracking shall be understood as something other than the ability to find multiple routes at once, either for different destinations or for the same destination. It stands for an ability to track simultaneously paths that would require a vertex to be able to have multiple

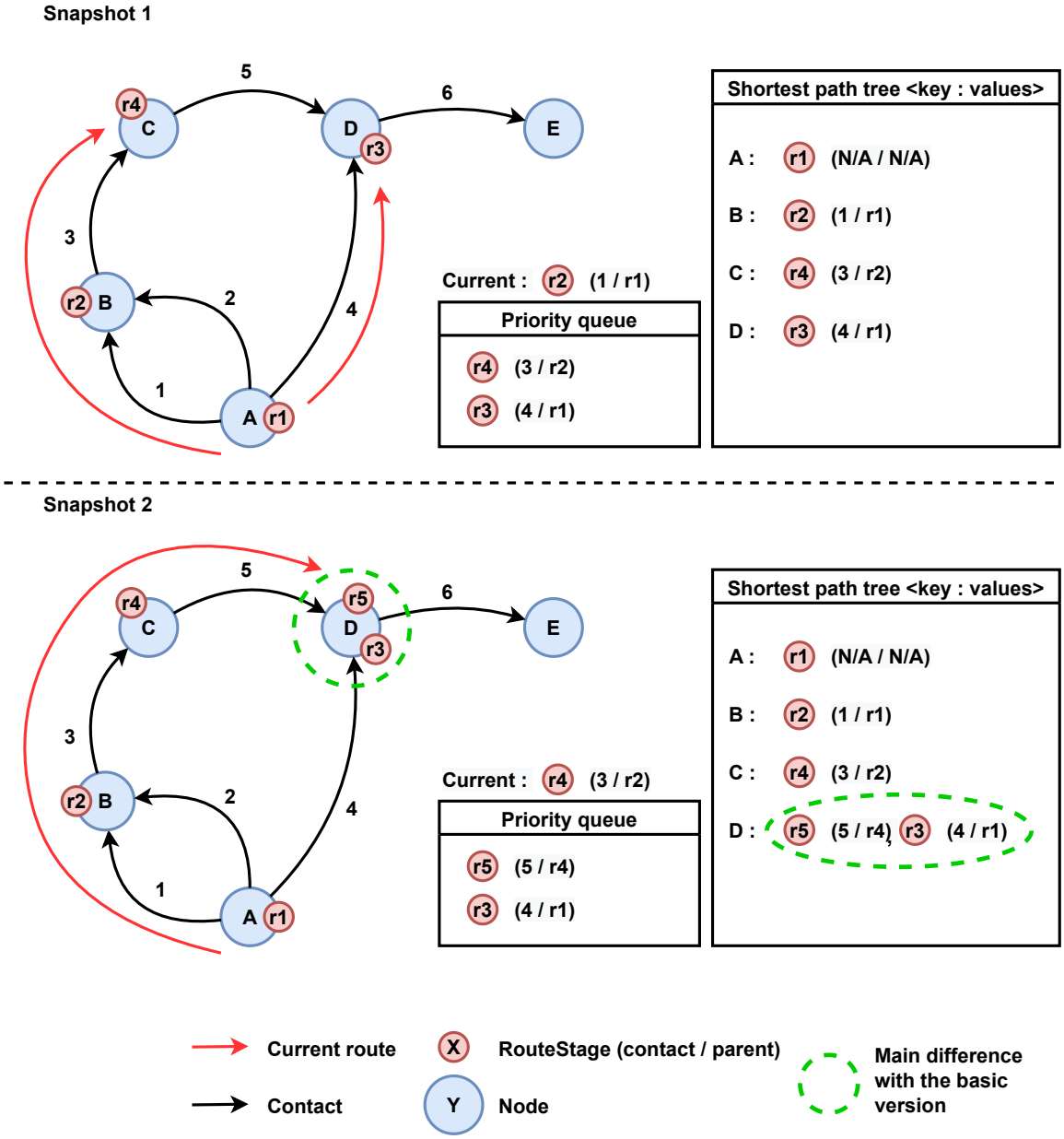


Figure 6.4: State snapshots of the multipath tracking tree construction.

parents. Considering figure 6.3, if the best route to node *E* leveraged *r*3 instead of *r*5 (e.g., because contact 6 starts late enough), then the shortest path tree shall track *r*3 and *r*5 simultaneously, as *r*5 encodes the best path to *D* and *r*3 is a dependency to the best path to *E*.

Consequently, the shortest path tree now holds, for each destination as a key, a list of RouteStages instead of a single one. This explains the MPT depth third dimension from figure 6.2. A comparison for the scenario depicted in figure 4.2 is proposed in table 6.1.

The implementation of this strategy is depicted in figure 6.4 as the multipath tracking versions of the snapshots taken in figure 6.3. The RouteStages are sorted by arrival times in the same way ($r1 < r2 < r4 < r5 < r3$). Snapshot 1 is identical from one figure to another

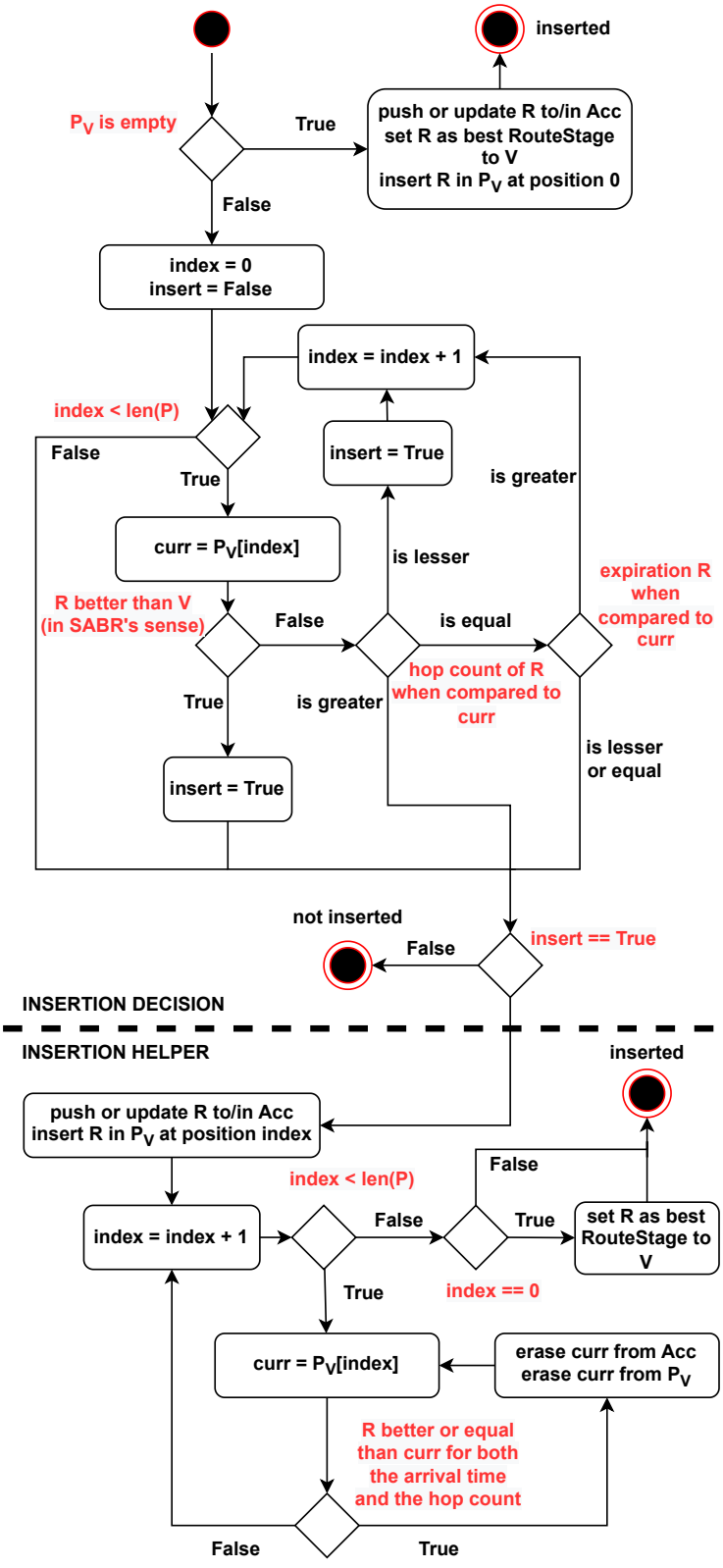


Figure 6.5: Insertion mechanism of multipath tracking.

because no multipath tracking capabilities are required for this specific iteration. However, snapshot 2 shows the behavior mentioned above by appending $r5$ to the list of RouteStage reaching node D.

It shall be noted that $r5$ is placed at the first position of this list rather than being appended at the end, as the list shall be sorted in the sense of SABR (considering the arrival time, hop count, and expiration). This ensures that the first element of a list is the best RouteStage to a node, the one leveraged for forwarding. Because $r3$ is not replaced, it is also not removed from the priority queue. This pathfinding technique can eventually detect the best route to E during the iteration, where $r3$ is the current element retrieved from the priority queue.

At this stage, it can be observed that this approach can be seen as a variation of a current contact retainment, as pushing several RouteStage leveraging different contacts to the same node without strict replacement within the shortest path tree map is analog to pushing contacts in the priority queue as processed in ION, but the analogy stops here.

A notable difference is an ability to ignore contacts thanks to the sorting policy of the contacts between a pair of nodes but to mitigate greedy behaviors drastically. It is important to detect whether a candidate deserves to be appended to a RouteStage list. To this end, the RouteStages sorting is leveraged.

Sorting is maintained in a SABR-compliant manner, and the insertion mechanism works as follows:

1. The process iterates from the best to the worst element in the list and stops as soon as the correct index for the candidate is identified (prioritizing first earlier arrival times, followed by lower hop counts and later expiration times).

2. After insertion, the process iterates from the next element to the last element. For each RouteStage element, if the inserted candidate is better than this element for all considered metrics, this element is removed from this list (and from Dijkstra's priority queue).

Figure 6.5 depicts the workflow of this insertion. P_V is the list of routes to node V. R is a new candidate route leading to V. The accumulator Acc is Dijkstra's priority queue. The insertion can be divided into two steps. First, the right index for the new candidate shall be found in the current list of candidates for this node (sorted in a SABR-compliant manner). If, during the search for this index, a known candidate appears to be better for all metrics when compared to the new candidate, the new candidate can be discarded, and the procedure stops. Otherwise, the candidate is inserted at the right index, and the second step starts. This second phase checks the next candidates in the list to detect if the newly inserted candidate can justify the deletion of previously found candidates.

At the end of the computation, the container T can be returned the same way it is returned without multipath tracking. The first element of each list is the best RouteStage for a destination and can be used for path retrieval from the reverse path starting at this RouteStage.

Hop tree construction

As an alternative to the basic and multipath tracking constructions, the hop tree construction is also provided to reduce the path lengths based on hop counts.

This modification provides a shortest-path tree version of the CGR-hop approach sketched in [RFM+21]. In the same way, this approach can be leveraged for congested environments.

With this flavor, a single RouteStage is retained at each node.

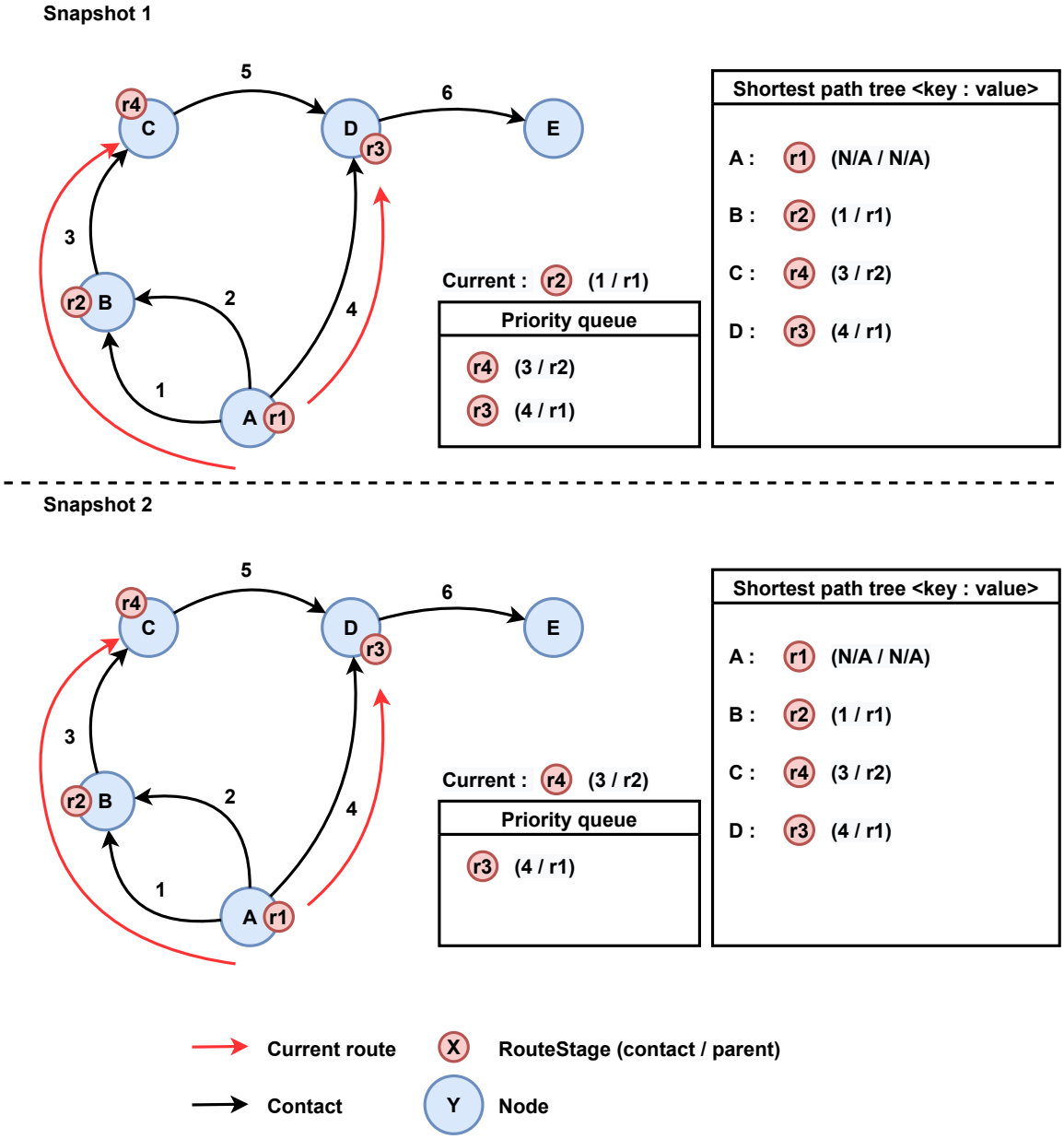


Figure 6.6: State snapshots of the hop tree construction.

If considering figure 6.6, snapshot 1 would again be identical if compared with figures 6.3 and 6.4. However, the RouteStage $r5$ would never get the chance to be inserted in the tree or the priority queue, as depicted in snapshot 2. Indeed, the existing RouteStage $r3$ leading to D is strictly shorter in hops, discarding $r5$ before any insertion. In the next iteration, $r3$ will become the current element, and the final path to E will be identical to the one that would be found with multipath tracking. However, the path leading to D would remain suboptimal in the sense of SABR, as the RouteStage $r5$ leading to D (which was discarded) is strictly shorter regarding the arrival time.

6.2.3 Tree management

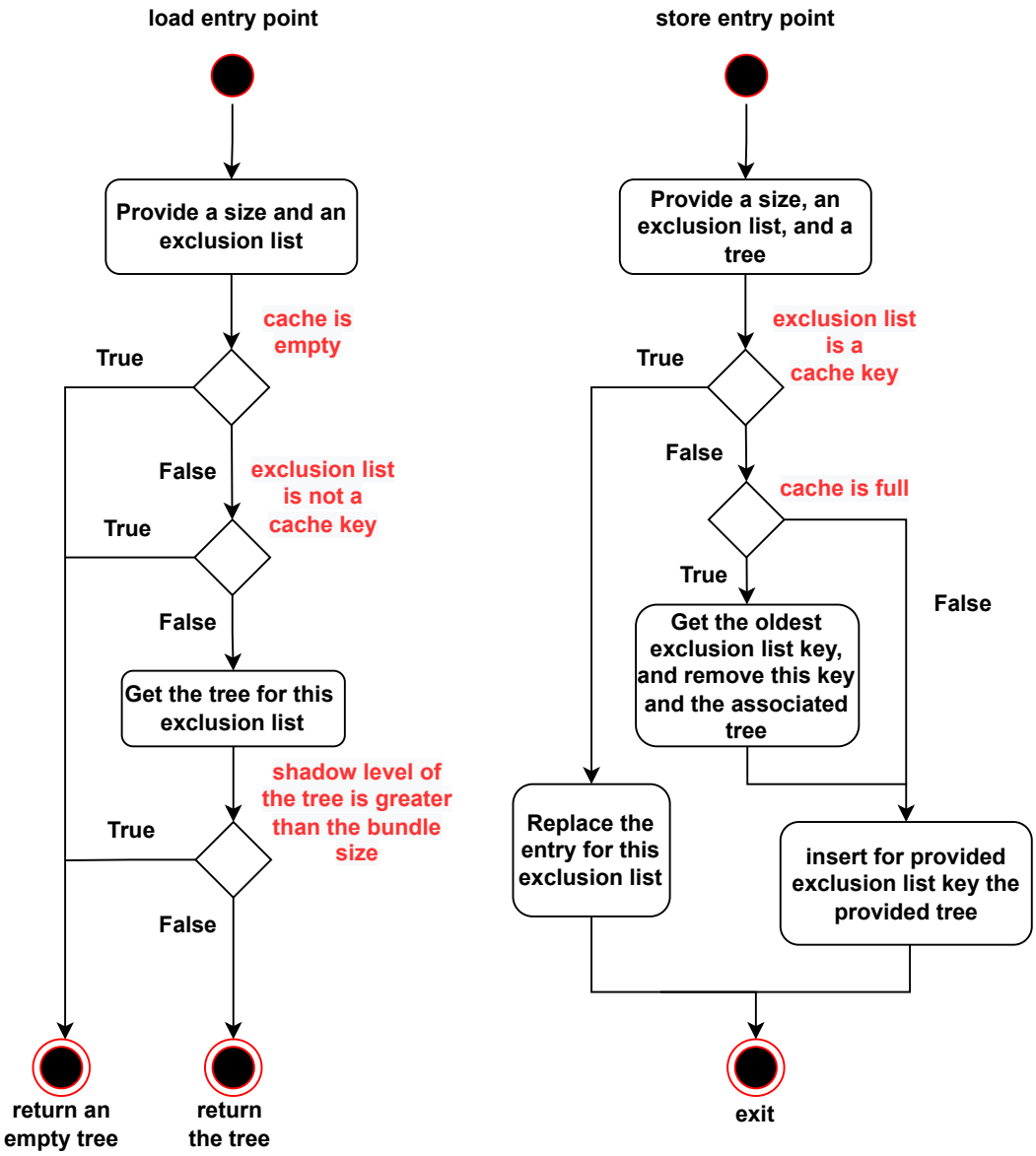


Figure 6.7: Caching mechanism.

The tree management component is the entry point for routing purposes. Given a bundle to schedule, this component will process requests to the cache and use the underlying volume management functions for dry runs and volume booking updates.

A prefiltering is conducted to verify that failure to schedule a similar bundle did not occur in the past. Otherwise, the bundle can be ignored right away.

Listing 6.1: Python pseudo code for dispatching map generation.

```
dispatching_map = {}
for dest in remaing_destinations:
    if target_route.to_node == dest:
        continue
    else:
        next_route = target_route.children[dest]
        if next_route not in dispatching_map:
            dispatching_map[next_route] = [dest]
        else:
            dispatching_map[next_route].append(dest)
```

Then, the cache is accessed to request a tree, with the bundle characteristics as arguments. If a tree is returned, the scheduling attempt function is called, and the procedure stops if the call is successful. The first hop contact and the list of booked intervals are returned. Overbooking can also be supported at this step. If applicable, the list of bundles that must be rescheduled is also returned.

The scheduling function relies on two recursive sub-functions. Recursivity is particularly practical for trees, especially for multicast bundles. In SPSN, unicast and multicast bundles are scheduled with the same procedures.

Due to this aspect, construction of the path from the reverse path for each destination is insufficient, and a dispatching map is required (creation of the dispatching map is described in listing 6.1). Thanks to the dispatching map, the dry run and booking sub-functions can be called one after another. If the tree is fresh, the dry run can be omitted.

The dry run recursive function includes calls to the volume manager for single-hop dry runs on individual contacts along the path. The recursive booking function contains calls to the volume managers for volume updates on those same individual contacts carried by the RouteStages constituting the tree.

6.2.4 Tree caching

For a given set of constraints associated with a bundle, the cache shall provide a suitable tree that shows compatible characteristics. Compatibility is detected thanks to the constraints applied to the tree construction and stored alongside the tree.

If the cache fails to provide a tree, a new tree shall be computed, and each time a tree is computed, an attempt to store the tree in the cache is processed.

Figure 6.7 depicts the current mechanism leveraged in SPSN. The caching strategy is node exclusion-based, i.e., at most, one tree per exclusion set is retained: a tree is eligible to be used by all destinations sharing the same exclusion set.

The motivation of the node exclusion-based strategy is to maintain those performance in a congested environment, where a single tree for all destinations would trigger the recomputation of the tree if the bundle to schedule shows a different exclusion set to the previous one.

Another motivation is volume booking conflicts that can occur in the multicast case. Proceeding to one scheduling per exclusion or using basic tree construction can help address those issues.

6.3 Contact segmentation

This section is divided into three subsections. Firstly, a general proposal for volume management will be described to render any routing algorithm compatible with any volume management technique if the latter implements a generic volume management interface and if the former is designed to operate with this interface.

Then, contact segmentation will be covered, with two different flavors, the simple volume manager and the enhanced volume manager.

6.3.1 Volume management interface

Volume management is operated with generic managing components attached to the contact (one per contact). Using an abstract interface permits the routing algorithm to work with any volume management technique implementing the interface.

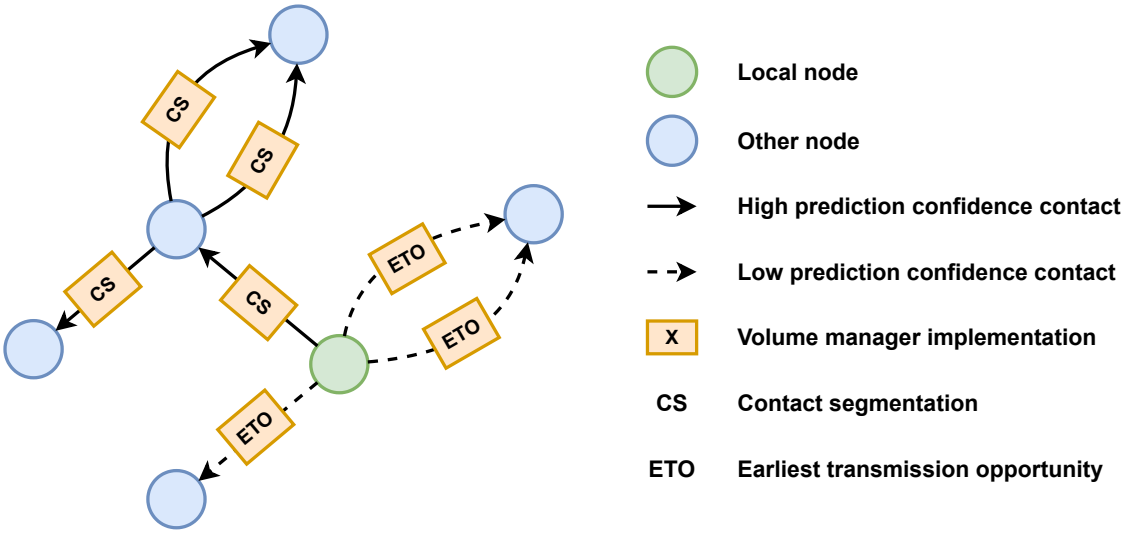


Figure 6.8: Deployment of heterogeneous volume management techniques implementing the abstract volume manager interface.

In SPSN, the RouteStages act as dry-run objects and are supplied to the volume manager for operations. This choice is conceptually and performance-wise adapted to single-hop management techniques. The schedule functions output the booked interval: after booking, SPSN stores on the bundle object for each contact it is planned to traverse the planned transmission start time, as well as a reference to the volume manager of the contact, to ease potential unscheduling.

Because SPSN proceeds to capacity-oriented tree construction, the dry run function is leveraged during pathfinding and tree management. If the tree is fresh, the dry runs processed during tree construction can be leveraged directly for scheduling, and the dry runs within tree management can be skipped.

The abstract manager can be an interesting tool for complex networks with heterogenous nodes and links. Nodes with sufficiently high processing power can attach to the contact objects more complex managers, while resource constraint nodes can leverage simpler managing techniques. It can be interesting to attach different managers to different contacts

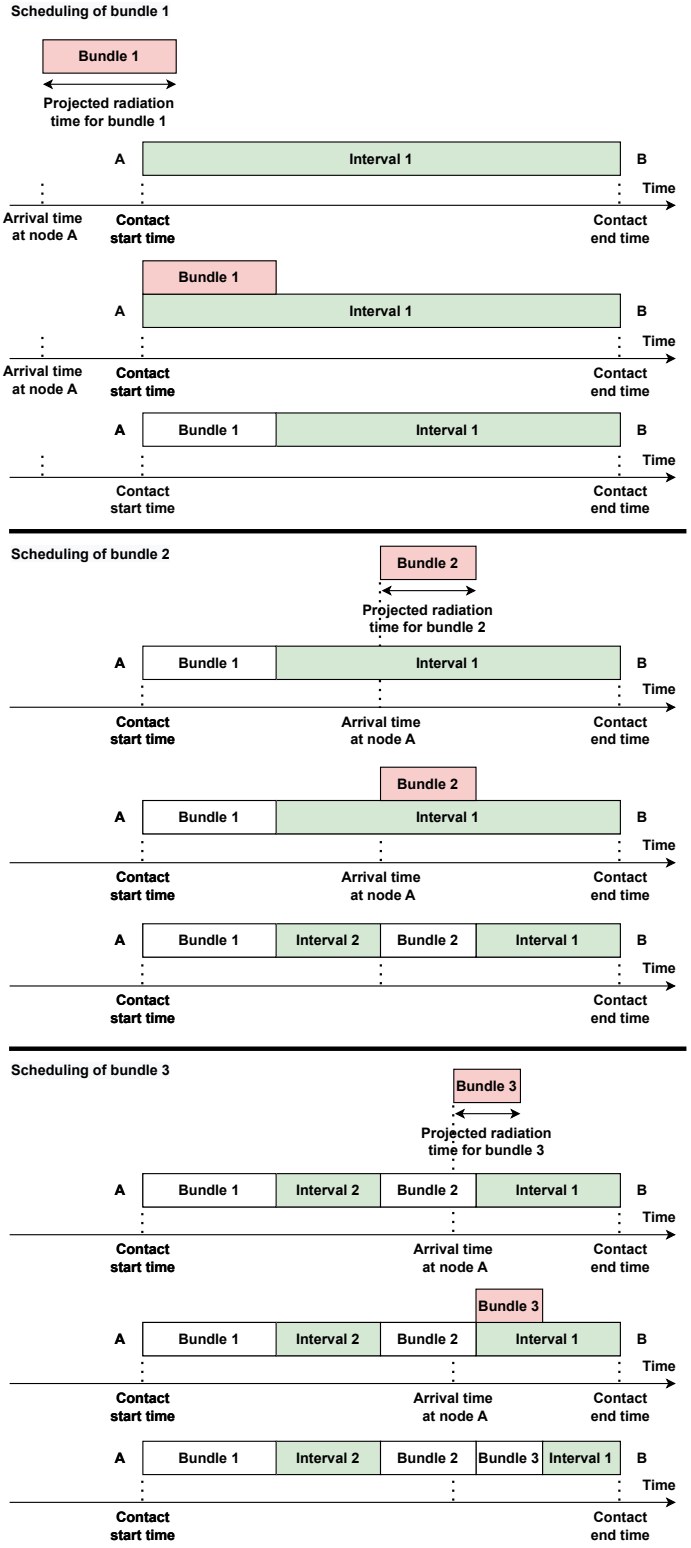


Figure 6.9: Segment management upon three subsequent bundle scheduling on a contact between node *A* and node *B*.

on a single-node basis. A static wired link, for example, would not need the same data rate variation support as a ground-to-LEO satellite link with its characteristic bell shape data rate variations over time.

Decoupling pathfinding and volume management would be valuable to the field for research and development concerns: moving away from monolithic routing algorithm implementation lowers the software complexity and simplify the development of particular concerns. A generic interface allowing different implementations is also suitable for evaluation purposes.

The volume managers allow, on the same node, to use different techniques for different contacts of the same contact plan. As shown in figure 6.8, some first-hop contacts use an ETO manager, while others use another technique (here, contact segmentation), still implementing the same interface.

6.3.2 Simple volume manager

In the base version, the contact list of intervals only carries the intervals of bandwidth availability. At contact initialization, a single interval is present in the list, with start and end times equal to the contact start and end times. The dry runs select the best intervals that can be leveraged while the update procedures apply the changes.

Figure 6.9 depicts the subsequent scheduling of three bundles on a contact. The segments in white are stored on the concerned bundles. The remaining available segments (in green) are stored on the contact (within the volume manager). Considering interval 1, it can be observed that the name remains unchanged after being modified.

For the first bundle to be scheduled, the radiation time (transmission duration) is calculated with bundle size and the nominal data rate of the contact. The transmission start time is found by iterating over the given intervals of availability. As a single interval of availability is present, the transmission start time coincides with its start time because the bundle was injected before this time at node *A*. The interval is then stripped from this radiation interval, and the stripped interval (projected transmission start and end times) is stored on the bundle object.

For the second bundle, a single interval is available (interval 1 was stripped and not split), but the injection time of the bundle is later when compared with the interval start time. If such a situation occurs, the interval shall be split to retain an interval of availability starting at the former interval start time and ending at the predicted bundle transmission start time.

For the third bundle, the first available is not the first one in the list, and the injection occurs prior to this interval start time. The interval does not need to be split and is only stripped according to the projected transmission start and radiation times.

It can be observed that interval 1 is not strictly split (with, for example, the introduction of intervals 1.1 and 1.2) but is stripped from the left, and a new interval is inserted before it. This reflects the implementation mechanism of interval management, allowing to maintain coherence if RouteStages metrics hold specific intervals in addition to specific contacts.

A second observation is that the example merges dry run and volume update. In practice, dry run information can be stored in the manager object to avoid recalculations during an update process. Also, if an interval is too short, the next one is considered. If no interval is suitable, the dry run is considered unsuccessful, and no dry run information is stored on the manager object.

Thanks to the booked interval information stored in the bundle directly, unscheduling a

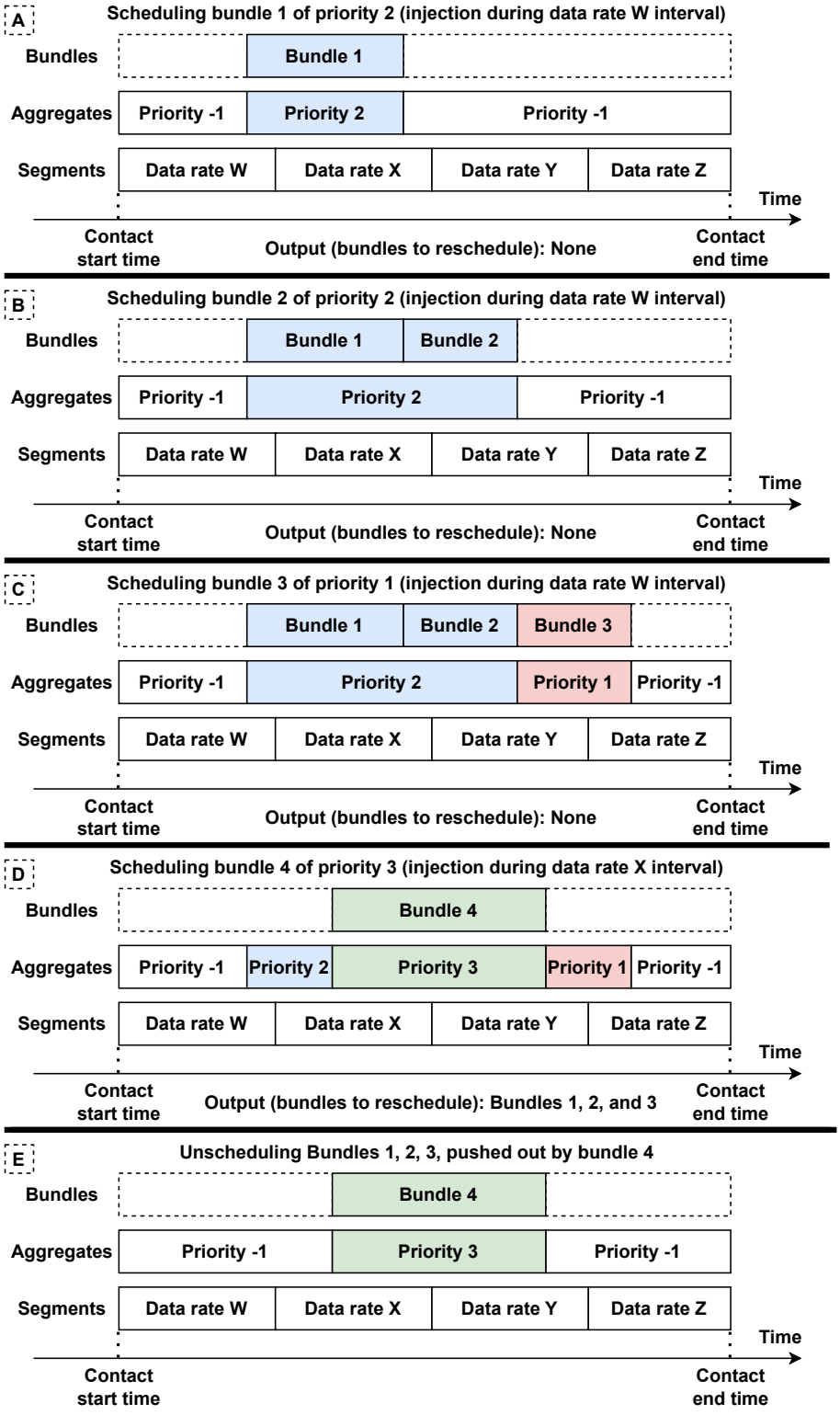


Figure 6.10: Volume management with varying data rates and bundle priority support. An interval is available if its priority is equal to -1.

bundle is processed by calling unscheduling procedures thanks to the planned transmission times and the references to the volume managers.

6.3.3 Enhanced volume manager

The simple volume manager is sufficient to support the bundle protocol version 7. This section proposes extensions to cover priority support and data rate variations for backward compatibility and finer-grained representation of the contact volumes.

In SABR, priority support is organized as virtual channels, one per priority level, and each channel shares the bandwidth with the higher priority channels (see section 3.2.4). This means that if a bundle of low priority is scheduled for an interval, the scheduling of a higher priority bundle for the same period can claim the same interval even though this interval is not available.

It is proposed here to detect the bundles that must be rescheduled when such a reclaim occurs.

Simultaneously, data rate variation shall be supported, as if the contact fragmentation was processed, but without drawbacks relative to the support of non-fragmentable bundles. A contact can be fragmented into smaller contiguous contacts of different data rates. Support of transmission overlapping several contacts would have to be managed on a top layer (overlapping two or more contacts). It is proposed here to support this feature on the contact layer.

To this end, the manager data structure shows:

- A list of segments for each data rate considered (this list is expected to be static).
- A list of aggregates[1], carrying the intervals of bandwidth of availability or bandwidth utilization for specific levels of priority.
- For each aggregate, a list of intervals tracking for each bundle scheduled its start and end transmission time.

At the initialization, a single aggregate of bandwidth availability with start and end times equal to the contact start and end times is created.

When a bundle must be scheduled, it can claim a bandwidth interval overlapping one or more aggregates of lower priority or available bandwidth. If some aggregates of equal or higher priority exist, the bundle can only be scheduled after their termination. Each time an aggregate is claimed or released, the algorithm can split or merge existing aggregates. When an interval is claimed, the bundles that overlap the claimed interval are returned for rescheduling.

The transmission start time is trivial to calculate. The transmission end time shall be calculated thanks to the data rate segments. The transmission start and end times can be on different data rate segments.

Figure 6.10 proposes a complete scenario implementing this system. Scheduling of bundle 4 is a single step. For convenience, it is split into two steps (*D* and *E*).

[1] Special thanks to Scott C. Burleigh for his contribution by suggesting the use of aggregates.

6.4 Contact passageways

The concept of contact passageway will be covered in this section. The first subsection will discuss the nature of the passageways with a new border definition. In a second subsection, the concept of virtual nodes will be depicted. Lastly, a third subsection will cover administration and pathfinding.

6.4.1 Regional border definition

With contact passageways, the nodes are no longer passageways on a conceptual layer, and the border is shifted to contacts between two regions. A node casually operates as a border between two regions when transmitting or receiving a bundle via a passageway contact, but it is not part of the two regions simultaneously.

This brings high flexibility by allowing the nodes to act dynamically as border nodes thanks to pathfinding and the set of contacts defined as passageway contacts rather than static configuration. This also means no nodes are special anymore and can share the same configuration.

Forwarding to a neighbor region is therefore defined as the shortest path finding to the neighbor region, i.e., the last hop contact is a contact passageway to this region. Thus, a bundle destination shall first be translated before a scheduling attempt. If the destination is part of the local region, the destination remains unchanged. If the destination is interregional, the destination is translated to a neighbor region (see section 6.4.3 for details).

A subject of criticism could be the increase in the number of nodes in the graph if the number of contact passageways (with different receivers) increases. The concept of virtual nodes introduced in the next section addresses this issue.

A wider scope permits better pathfinding, and the absence of nodes aware of the two region contact plans could be seen as a weakness. However, there is no good reason to think that the passageway contact would not leverage the best path in most cases, and the following argument can support this. If multiple-node passageway support is available, a non-passageway node that needs to forward a bundle to a neighbor region will choose the fastest path leading to any passageway node. This would be the same behavior as with contact passageways. However, providing an optimal mapping between contact and node passageways design is complicated.

Even if a wider scope could effectively increase the performance, all the nodes involved in the passageways contacts between two regions would be required to constitute an interface region. This, however, includes the addition of an extra region.

Adding regions makes mapping between the two approaches complicated, if not impossible. More importantly, it would render the node passageway design hardly scalable and harder to administrate if the region count increases drastically, as one interface region would need to be added to optimize each regional neighbor-to-neighbor forwarding.

Also, the destination regional membership is an input for destination translation, and the output is a neighbor region. The following assumptions shall therefore be considered:

- A node knows the regional membership of an interregional bundle.
- A node can infer the next regional hop from a destination region.

Such information can be retrieved thanks to local information or thanks to the information provided by the bundle (in an extension block). The administration will be discussed in section 6.4.3.

Support for shortcuts turns the tree into a mesh (see figure 2.1). This thesis still refers to a tree, as the tree approach might be helpful, at least for region naming (see section 3.1.6). Also, if two paths appear to have the same distance (see section 6.4.3), a path encompassing fewer hops leveraging a shortcut could be preferred if the trust in the original tree structure is considered higher than the trust in the possible shortcuts. In the same way, after the detection of a loop, hops using the original tree structure could be preferred. Those concerns will be considered in future work.

6.4.2 Virtual nodes

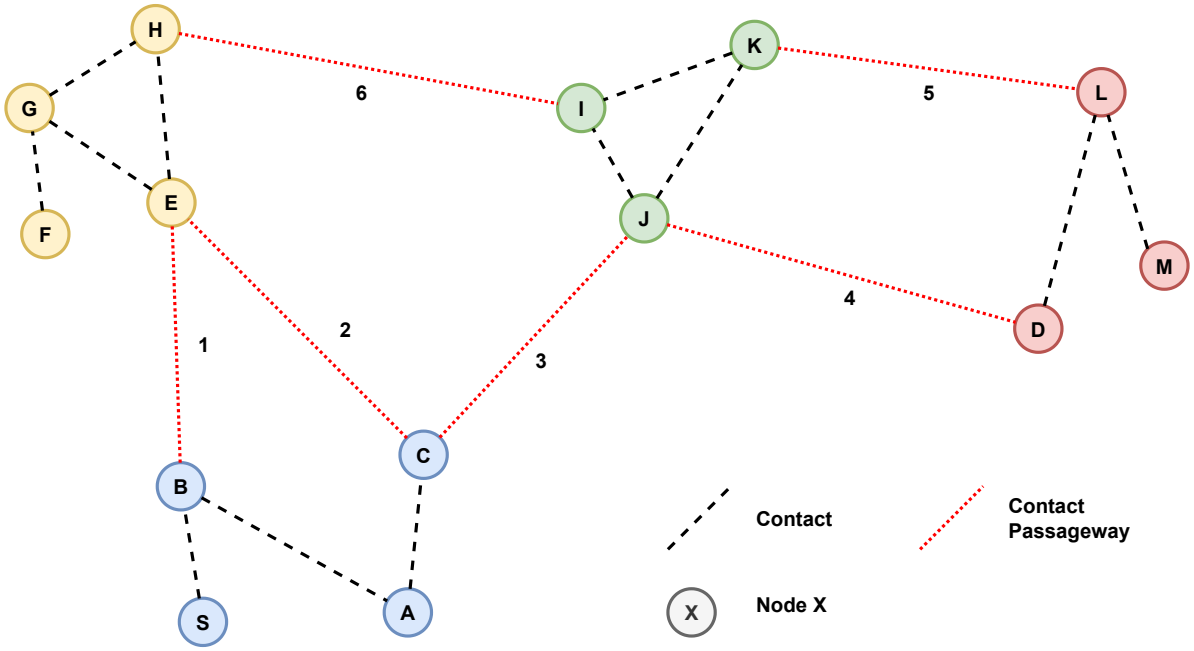


Figure 6.11: First step to create a global representation: choose the passageway contacts.

If the passageway contact count between two regions is high, a non-negligible number of new nodes may appear in the home region contact plan. It is proposed here to leverage vertex contraction to create a virtual node resulting from merging all nodes that are members of a given neighbor region. This way, the network graph representing the region would present a single extra vertex per neighbor region to avoid the overhead of adding a vertex for each egress passageway contact receiver.

The passageway contacts, therefore, present a virtual receiver or a virtual transmitter node. No interregional router exists, as a destination is translated to an intraregional destination, an actual node, or a virtual node representing a neighbor region. For an effective application of the (intraregional) router, the contact objects shall present at least an extra attribute: the actual receiver in addition to the standard one. If the receiver is a virtual node and the local node is the sender of the contact passageway, a real node ID is required for enqueuing. Consequently, the standard or virtual receiver is considered for pathfinding, and

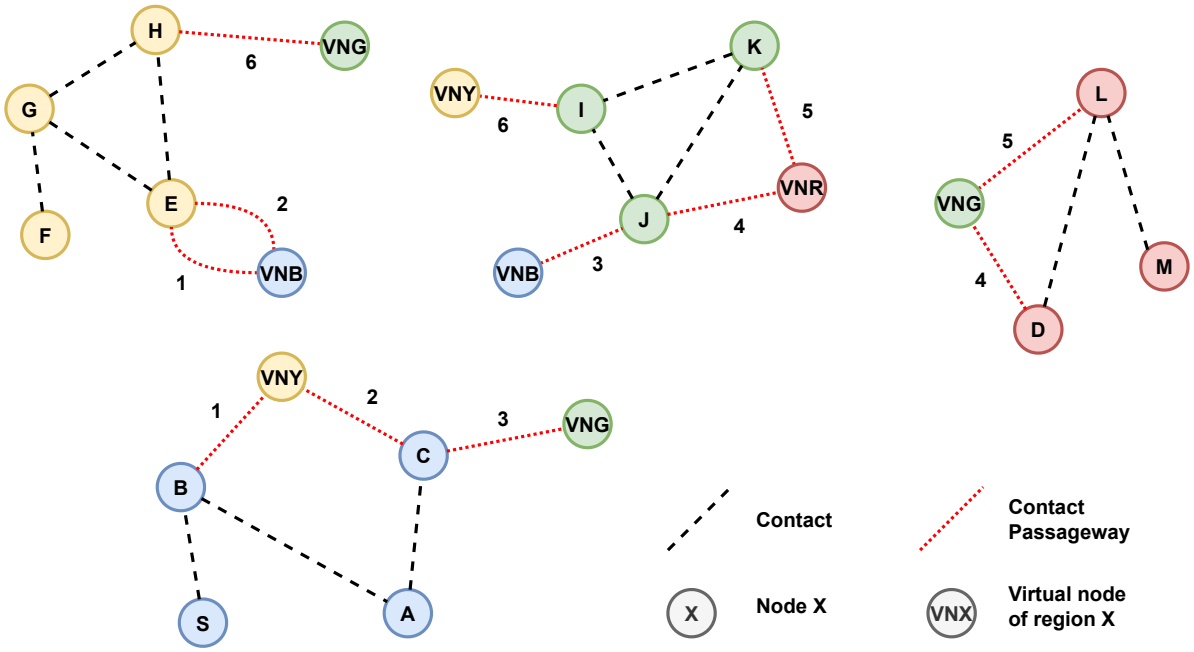


Figure 6.12: Second step to create a global representation: apply vertex contraction on the nodes that belong to the same neighbor region.

the real one is considered for enqueuing.

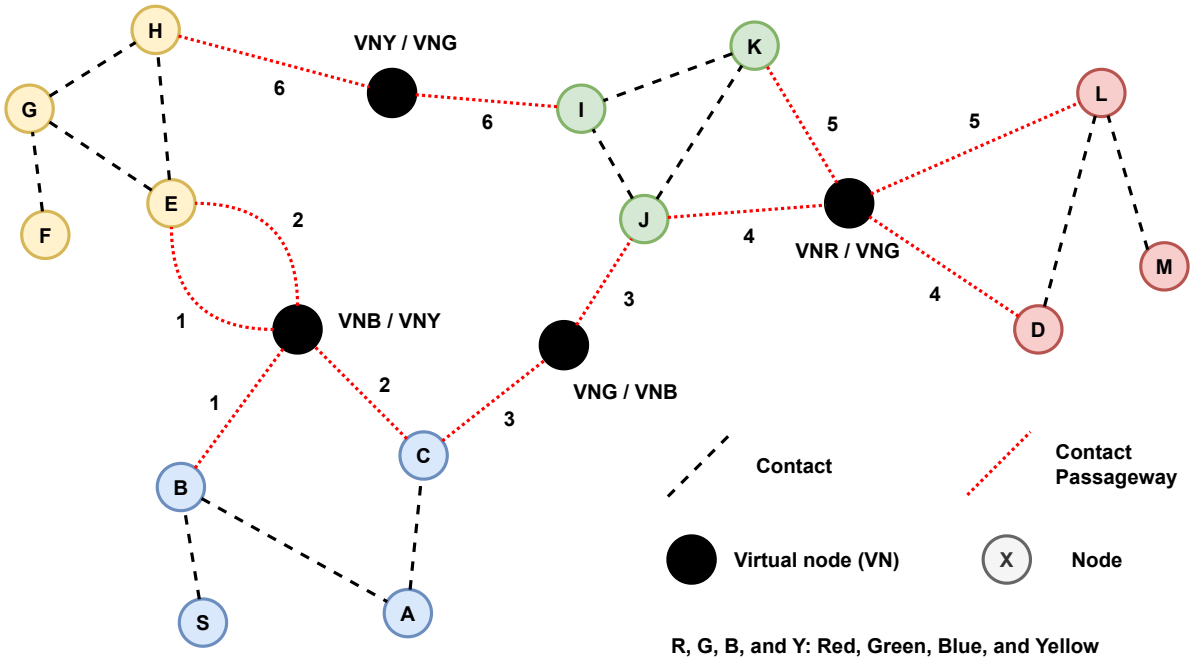


Figure 6.13: Third step to create a global representation: if two regions are neighbors, apply vertex contraction on the dedicated virtual nodes.

Creating a connected global representation of such a network (while still representing the virtual nodes) requires three steps. The first step is to assign a region to each node and elect

the contact passageways, as depicted in figure 6.11. Regional membership is represented with colors.

The second step allows the creation of intraregional topology representations as seen locally by the regions. Those graph representations are the ones on which SPSN operates. Each local region node gathers all contacts of the contact plan to create the graph (vertices for nodes, edges for contacts). The presence of contact passageways allows the creation of vertices (nodes) that do not belong to the local region (senders or receivers of contact passageways). Vertex contraction is then applied to all the nodes belonging to the same neighbor region. This allows the presence of a single virtual node per neighbor region. The resulting graph is depicted in figure 6.12.

Discussing an interregional structure with a disconnected graph as a support can be considered unintuitive and impractical. The third step can then be leveraged to create a connected graph while still representing virtual nodes. This is possible by applying vertex contraction on the virtual nodes of regions that are neighbors. Consequently, the passageway contacts are represented twice (on both ends of a merged virtual node). The resulting representation is depicted in figure 6.13. This representation highlights that a node can be a border node for two neighbor regions (there are no actual limitations) without the need to support three contact plans (node *C*).

With the virtual nodes implemented, pathfinding becomes trivial. For a given interregional bundle, the destination is translated to a neighbor region represented by a virtual node, and the bundle can be scheduled with the corresponding virtual node as the destination.

The contact plan format is considered out of scope. Regional membership should be provided alongside the contact plan for each node present in the plan and can be updated during the operational period. The graph initialization can be adapted to create the virtual nodes accordingly. There is, however, one behavior that needs to be handled: contacts 1 and 2 can overlap in time, which is prohibited by SABR.

Overlapping cannot be considered a problem. The first contact (earliest start time) will be leveraged, and the performance should remain acceptable. However, capacity-oriented search does not protect against issues that arise from overlapping contacts showing different data rates, delays, or booked volumes. Consequently, a contact that would normally be ignored can show a better arrival time than the selected one.

This overlapping issue cannot be considered a challenge and is handled in SPSN. Apart from this use case, it could ease contact planning to discard this SABR limitation. It could also be interesting to be able to choose a specific interface for a bundle, but the benefits are highly speculative and far beyond the scope of this thesis.

6.4.3 Pathfinding and administration

As this approach is region-based, the nodes shall know to which neighbor region a bundle shall be forwarded rather than to which specific border node. The source knows the regional membership of the destination, but the other nodes might not. Bundle encapsulation [Bur18] or extension block can be leveraged to either get a bundle with a destination region as the bundle destination or attach the destination region to the bundle in an extension block. The former would require discussions about naming and addressing. Details about the final handling decisions are considered future work, and the final region will be termed attached to the bundle for convenience.

Also, the approach's effective application relies on the component's good functioning that

translates interregional destinations to intraregional virtual nodes, presupposing the existence of a global or partial knowledge of the regional structure within this component. Two approaches are proposed: static and dynamic configurations.

With static configuration, each node stores locally the regional membership of the nodes (destination) they are interested in and a set of regional neighboring information to construct the regional graph. The graph is leveraged to build a forwarding table thanks to Dijkstra's shortest path tree algorithm each time the regional neighboring is updated. When a bundle is forwarded, translation occurs, and the destination region is attached to the bundle. The regional forwarding table is then leveraged to forward the bundle to the next-hop neighbor region (by finding the shortest path to the correct virtual node). As all nodes in the network know the regional structure and the destination region is attached to the bundle, each node along the path can proceed to forward until the destination region is reached.

With dynamic configuration, the nodes are not aware of the regional membership of the nodes in other regions. They are only aware of regions that are neighbors to the local region (they see the virtual nodes representing those regions in the local graph representation). When a bundle has a destination not part of the local region and not yet known by the translating component, probes are sent to each neighboring region. The probe is an extension block attached to the bundle. The nodes traversed by the probe bundle shall add to the extension block their regional membership if not present (in a stacked manner). When the bundle reaches the local region of the destination, backpropagation occurs thanks to the path encoded in the probe, sending back to the source a message carrying the regional path that allowed the bundle to reach the destination. The source may receive more than one response. Each response allows the source to update its local view of the global region neighboring to maintain a regional graph. Thanks to this knowledge, the forwarding table is computed with the Dijkstra algorithm. The forwarding table is used within the translating component for the next bundles sharing the same destination.

Considering figure 6.13, and an interregional bundle that has to be sent from *S* to *D*, probing works as follows:

- In the source region *blue*, *S* sees two virtual nodes. The bundle is sent as two copies to those two virtual nodes with a probing extension block attached, probe 1 to *VNB*/*VNY* and probe 2 to *VNG*/*VNB*. Both probes carry a single value, *RB* (for region *blue*).

- In region *yellow*, probe 2 is received via *E*. Node *E* detects that the final destination node is not part of the region. Probing shall be pursued. As *RY* is not part of the extension block, this value is appended to the block, and the bundle is forwarded to the sole virtual node representing a region not part of the extension block, *VNY*/*VNG*. Nodes of the region *yellow* along the path leave the extension block untouched, as *RY* was already inserted by node *E*.

- In region *green*, 2 probes are received, probe 1 from region *blue* (block values: *RB*) and probe 2 from region *yellow* (block values: *RB*, *RY*), by nodes *I* and *J*. Both nodes append *RG* to the probe blocks, and the probes are forwarded to the single region that is not part of the block, the region *red*, i.e., to virtual node *VNR*/*VNG*.

- In region *red*, 2 probes are received from region *green*, probe 1 (block values: *RB*, *RG*) and probe 2 (block values: *RB*, *RY*, *RG*). The receiving nodes in region *red* detect that destination is part of the local region. Backpropagation occurs with a response message (one per probe) using the reverse path encoded in the received probe (could be

sent via the shorter path if more regional neighboring information is available locally). The response messages carry the path the probes took from the *S* to the local region *red*.

- Node *S* receives the 2 probe responses, one with the values *RB*, *RG*, *RR*, the other with the values *RB*, *RY*, *RG*, *RR*. Those two messages are stored persistently. From the first response, the neighboring between *green* and *red* regions can be inferred. The second also informs about this neighboring, but also about the neighboring of regions *yellow* and *green*. With all those neighboring information, a graph can be constructed, and Dijkstra can be applied for pathfinding.

The probing strategy is similar to the one used for the state-of-the-art design depicted in section 3.1.6. The difference is that the probes are forwarded to the other passageways in the region rather than being forwarded to virtual nodes. The state-of-the-art design does not present shortcuts. The regional structure is a tree and not a mesh, discarding the possible presence of loops during probing. Consequently, the interregional path stack present in the extension block and the preliminary checking to avoid the insertion of duplicates are mechanisms only relevant to the design presented in this thesis.

Please note that this approach allows probing to build a graph to reach regions that were never discovered with direct probing for those regions. Indeed, paths to *yellow* and *green* regions are available, even if probing was only processed to find the membership of node *D* in the *red* region. Discussion about this behavior is considered future work. Also, the interregional interfaces are considered bi-directional. Evaluation of variations of this assumption is future work.

However, the knowledge inferred from the probing response is at the current step of the description, only available at the source. For another bundle that needs to be forwarded to *D*, the region *red* can be attached to it, but the next node after the source would not be able to find a path to region *red*. To cope with this issue, either the source shall attach the complete path to the bundle, or the backpropagation shall also inform all the nodes of each traversed region about the response (for example, with broadcast). The former approach would probably be preferred, but further investigation shall be conducted.

A combination of those two approaches could be used. It seems realistic to maintain on each node the regional structure with static configuration and occasional updates while dynamic destination regional membership and pathfinding discovery is leveraged. Also, instead of adding only a region id in the extension block, a node could add the local region and neighboring regions. This would also help the source to build its regional awareness.

In this first version, a static configuration is used. The interregional distance is considered to be the regional hop count. This approach is adapted for cases where contact passageways are selected as long-range links, e.g., for planetary regional structure. If the regions are not planetary-based, if contact planning includes intermittent shortcuts (with non-negligible delays between two periods of availability), or if shortcuts show expirations, the pathfinding distance should be adapted to support such topologies. In this case, the distance calculation would be analogous to the intraregional routing one. The interregional pathfinding itself would become dynamic. This aspect is beyond the currently considered future work in the context of this thesis. Using Dijkstra already allows the support for shortcuts, turning the tree into a mesh.

Last but not least, all nodes in the network can be part of a single region, and all nodes of a region share the extra same contact plan. This could ease administration if regions are administrated, for example, by different agencies, but further investigation shall be conducted

on this topic. The operators only have to agree on the passageway contacts, while the remaining links of a given contact plan can remain private. Compared to the node passageway design, increasing the size of a home region in terms of nodes and contacts would not damage the performance (node passageways are affected by the growth of any contact plan they carry). This holds even if new contact passageways are introduced: the virtual node depicting a local region remains unique in the contact plan of the neighbor region.

6.5 Integration

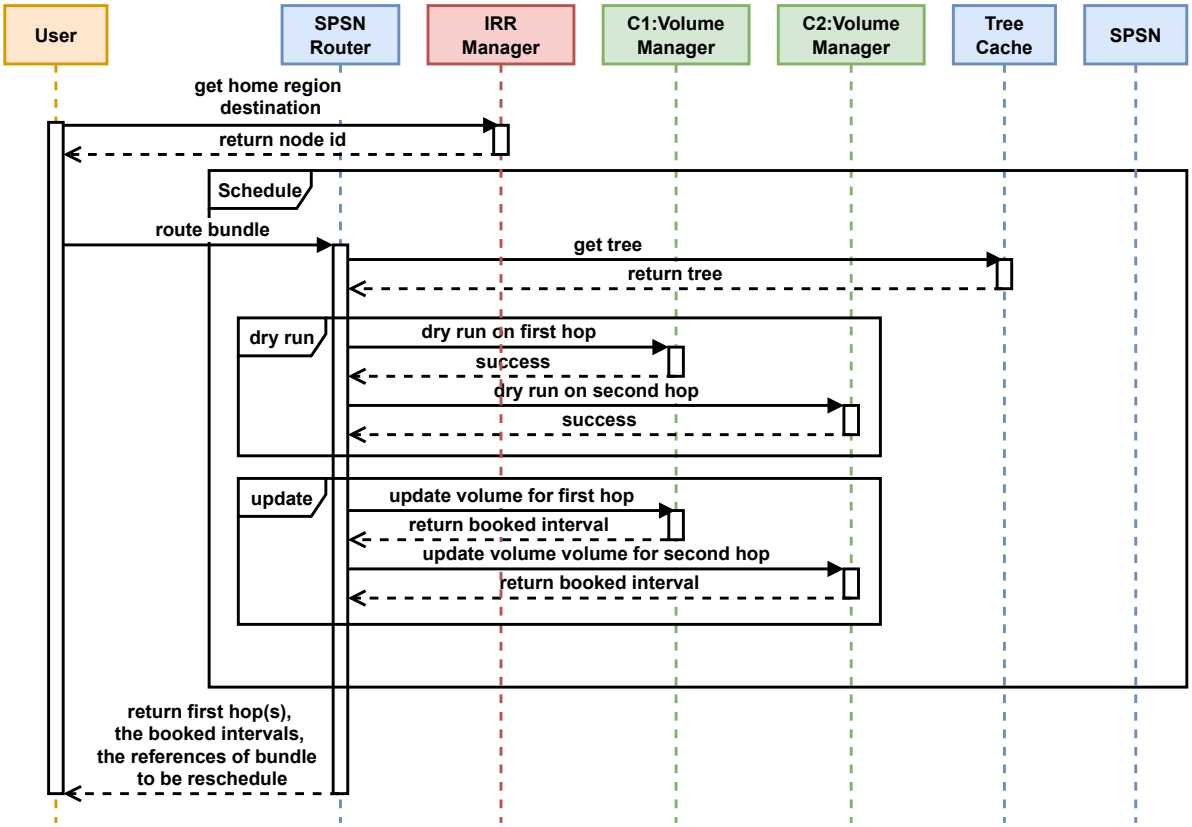


Figure 6.14: Sequence diagram for the interregional bundle scheduling workflow with volume management.

SPSN

SPSN was designed as an approach that could be provided as a routing service operated separately from the further networking stack. The current design allows the creation of an SPSN component by supplying contacts in an SPSN-specific format. A simple wrapper to convert contacts is therefore required. As a research implementation, this is not considered an issue, and support of other formats could be introduced later in a production-ready version of SPSN.

The information required to revert the volume booking of a previously scheduled bundle is, at the moment, not stored within the engine and shall be stored in the wrapper bridging

the utilizing DTN software and SPSN. This decision is practical if the DTN software cannot notify that a bundle has been successfully transmitted. Such information could be stored in SPSN directly if callbacks to SPSN are processed upon transmission success of a given bundle. In the same way, expired contacts are pruned automatically. Callbacks could also be leveraged to deal with such concerns.

Eventually, a production-ready implementation shall be compatible with a standardized DTN routing API (e.g., [Wal22]). Adapting SPSN to such API is relatively easy due to the already high independence of the current SPSN research implementation.

Contact segmentation

Contact segmentation is supplied with three main components. The first supplied component is an abstract volume manager class, and the inheritance of this manager (i.e., correct implementation of this interface) renders SPSN compatible with the volume management method. The two other components are two contact segmentation flavors supplied as volume managers implementing this interface to be easily exchangeable within SPSN.

Contact segmentation is compatible with CGR. However, deploying this approach is more complicated if CGR is incompatible with the interface. Thee interactions with the volume manager occur during route construction and route selection, requiring modifications in the inner parts of the targetted routing algorithm.

Contact passageway

The contact passageway concept deployment also requires specific considerations. First, the routing algorithm shall be compatible with the concept. SPSN would be a suitable candidate. Then, the routing engine shall be initialized with extra configuration, identical for any node sharing membership to a region. At last, the administration shall be adapted to be region-focused and not destination-focused by creating destination-based forwarding information on top of a regional structure. To this end, regional membership knowledge is important. The administration messages are expected to be rare in a quasi-static hierarchical design. In the same way, the administration of the multicast membership is not covered, as its required administration is identical to CGR-multicast in ION.

Workflow

To summarize the interdependency between the components, a complete workflow example is proposed in figure 6.14, depicting the scheduling of an interregional bundle. At the entry point of this workflow, the cache presents a suitable tree for this bundle characteristics, showing a two hops path to the virtual node of a neighbor region. The term InterRegional Routing (IRR) is used for convenience.

The *User* depicts any higher layer component leveraging SPSN for routing purposes. The *SPSN Router* is the entry point for scheduling. The *IRR Manager* cares about maintaining the interregional forwarding table. In this version, the *IRR Manager* is not accessed within the *SPSN Router* but by the *User* directly (subject to changes). The *Tree Cache* and *SPSN* pathfinding (in blue) are internal components of the *SPSN Router*. *SPSN* pathfinding can support various tree construction strategies (basic, multipath tracking, and hop tree constructions). The *Volume Manager*, one per contact, implement the requested volume management technique.

Component	Function	Input	Action	Ouptut
IRR Manager	*get_irr_destination*	destination id	access the interregional forwarding table	next hop region
	update_neighboring	region id, ids of the new neighbors	update the graph and recompute the forwarding table	N/A
SPSN Router	*route*	list of destinations, bundle characteristics (priority, size, etc.)	create or load a tree, proceed to a dry run, proceed to volume updates	a list of destinations for each first hop, the list of update intervals, references of the bundles to reschedule
Volume Manager	*dry_run*	current time, bundle size, bundle priority, dry run object reference	proceed with a dry run, store in the dry run object the dry run details	success or failure
	schedule	bundle size, bundle priority, dry run object, bundle reference	leverage dry run information to update the volume state	N/A
	unschedule	planned transmission time, bundle size	unbook volume from a specified time for a specified volume	N/A
Cache	*load*	bundle characteristics (size, node exclusion list for the destinations, etc.)	select a tree respecting the provided constraints	the selected tree
	store	tree, computation constraints (node exclusion list, size, etc.)	stores the tree as a new entry, another entry can be deleted	N/A
SPSN	*compute*	source node, current time, bundle characteristics (priority, bundle size, etc.)	computes a tree suitable for the provided constraints	the computed tree

Table 6.2: Components and functions of the scalable schedule-aware bundle routing approach.

The *User* provides the node id to the *IRR Manager* to translate this id to a home region destination. The destination is not part of the home region and the next regional hop virtual node id is returned. The *User* can then request bundle scheduling with the translated node destination id, the current time, and the other bundle constraints (e.g., node exclusions and expiration).

The *SPSN Router* forwards those constraints to the *Tree Cache* to retrieve a suitable tree. The *SPSN* pathfinding component will not be involved as a tree is available. The returned tree was cached and is consequently not fresh so the dry run information generated during its construction (or the last previous bundle scheduling) was already leveraged and rendered invalid for this scheduling. The *SPSN Router* proceed to a dry run of the whole path against the tree thanks to the recursive functions, triggering a dry run call to the dedicated *Volume Manager* for each hop contact. The term *dry run* is defined in section 3.2.2.

The dry run is successful (destination reachable before expiration, segments can be updated safely), the *SPSN Router* can update the segments recursively along the tree branches but this time with calls to the booking procedure on each *Volume Manager* along the path.

The *SPSN Router* returns for each first hop contact (several first hops for multicast) the list of destinations downstream for multicast purposes. A single destination being the next region hop virtual node id is part of the list. The *SPSN Router* also returns the booked intervals. If the bundle has to be rescheduled, the *User* can revert the changes due to those intervals. Finally, the list of references to the bundles that need to be rescheduled is provided. If the *User* organized the storage of the booked intervals alongside the bundles, cancellation of the booking is trivial before rescheduling.

A non-exhaustive list of the components and function are provided in table 6.2.

6.6 Summary

SPSN, contact segmentation, and the contact passageway concepts were proposed as a coherent routing approach with multicast, volume awareness, and interregional support.

SPSN proposes to provide reuse for trees rather than single-path destinations and provides modifications to Dijkstra's algorithm to allow the creation of SABR-compliant trees

thanks to multipath tracking. In addition, SPSN is built to work with abstract volume managers attached to the contacts.

Two volume manager implementations were proposed to provide contact segmentation capabilities with two levels of feature support. Contact segmentation intends to accurately track the contact's bandwidth utilization by recording the available and transmission periods.

The contact passageway design allows failover, optimized delays, and load-balancing-like behavior for interregional forwarding. The ability of the design to allow any node to be in a single region can ease the configuration. The approach is also expected to have a negligible overhead: at most, one node per neighbor region is added to the home region graph. Single nodes can bridge multiple regions without contact plan overhead.

7 Evaluation

The evaluation mainly focuses on comparison with SABR. Consequently, the evaluation will compare the proposed concepts with different CGR flavors. This decision is motivated by the wide spectrum of CGR concerns that had to be covered and its predominance for existing and future space networks. This statement is confirmed by the CCSDS standardization of CGR and the future Mars communication architecture described in the IOAG report [ioa22], which defines SABR as an essential data structure in the service management process.

Firstly, the methodology will be described, with the simulation tools, algorithms, and scenarios that will be leveraged for evaluation.

The computational pressure will then be evaluated for single bundle scheduling and volume management on single contacts.

Preliminary simulations will follow to eliminate routing approaches showing discarding features to ease further evaluation. Also, retaining those algorithms for simulation with larger networks would complicate the evaluation due to their processing time.

An evaluation of the algorithms under congestion will then be proposed to highlight other possible discarding features that remained for the algorithm flavors that were not discarded during the preliminary simulations. The impact of those features will also be discussed to allow a better interpretation of the results. For instance, unjustified drops are hardly acceptable but can lower congestion to a point that can show misleading results.

7.1 Methodology

7.1.1 Simulation tools

The evaluation leverages the aiodtnsim simulator toolchain developed by Felix Walter. The framework includes the simulator, the tools to create realistic inputs for the simulator, and orchestration tools for result gathering and plot creation.

The aiodtnsim simulator[1] is developed in Python with asyncio and permits mono-threaded reproducible simulations by externalizing the creation of the bundle injection plans supplied as a configuration. The injection plan is a configuration file leveraged to create the bundles during simulation. A bundle injection plan entry encompasses a source node, a destination node, the bundle size, its creation time, and its expiration times.

[1] `https://gitlab.com/d3tn/aiodtnsim`

The simulator permits the development of new algorithms by leveraging the inheritance of base implementations for opportunistic approaches or SABR.

Alongside aiodtnsim, the package dtn-tvg-util[2] permits the generation of the bundle injection plan using a contact plan as input. This contact plan, also generated by dtn-tvg-util, can be created with a dtn-tvg-util-generated scenario as input.

Additionally, dtn-tvg-util provides the capability to create ring road scenarios (presented in section 2) by using Celestrak[3] data to get realistic trajectories for the orbiters and generate the according contacts between the nodes (satellites or ground stations).

This framework was initially described in [Wal20].

7.1.2 Simulator extensions

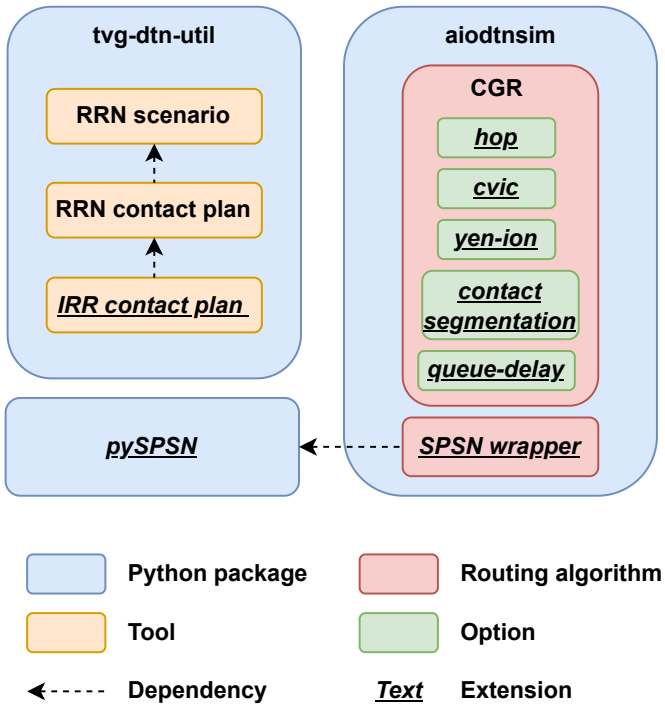


Figure 7.1: Contribution map to the existing components.

Figure 7.1 depicts the extensions required for evaluation. Modifications were introduced for aiodtnsim to embed SPSN as a wrapper using the pySPSN research implementation[4]. CGR was also modified to allow multicast and is now part of CGR's core (extension not represented in the figure). An alternative path search strategy was added to test the adaptive behavior of Yen's algorithm (option *ion-yen*). Regarding pathfinding itself, two options were introduced to either optimize the hop count (*hop*) or maintain a current contact instead of a current vertex (option *cvic*). The simple contact segmentation and the queue-delay approaches were also introduced for CGR, and the latter is used alongside the ETO for the first hop contact.

[2] https://gitlab.com/d3tn/dtn-tvg-util
[3] http://www.celestrak.com
[4] https://bitbucket.org/olivier-dj/pyspsn

Additional modifications were introduced to dtn-tvg-util, to allow the creation of interregional scenarios. This permits the creation of IRR trees compliant with the original approach (a single upper region for several lower regions). The regions are generated as ring road networks, bridged with long delay contacts. The realism of the scenario shows limits, as each region is generated as a realistic Earth ring road and not for specific other planets. However, this was convenient due to data access limitations and remains satisfactory for pathfinding evaluation.

The interregional routing approach does not mandate showing a binary tree structure. However, the systematic use of a binary tree structure will be practical in the evaluation context. Stating first that increasing the depth by one doubles the number of regions would be obvious. The binary tree has a second attractive property by showing a higher average path length if compared, for example, with a ternary tree. A binary tree of 15 regions would already have a higher depth than a ternary tree of 14 regions. The decision allows scaling up the scenario size in a controlled manner while increasing the topological complexity (here, the path lengths) to the maximum, stressing as much as possible the tested designs.

The interregional shortcuts are created but integrated into the simulated nodes' contact plans if shortcut support is enabled. As mapping between the node and contact passageway designs cannot be defined optimally, only the influence of shortcut support will be evaluated, alongside the good functioning of the proposed contact passageway design.

Additionally, a precise offline (i.e., not part of an aiodtnsim simulation) computational pressure evaluation will be conducted, with measurements on randomly generated contact plans, to be exempt from any load-dependent perturbation, and get precise per node and contact count pressure evaluation.

The volume management technique that supports all the optional features is expected to present some overhead. Therefore, the pathfinding offline evaluation shall be accompanied by an offline evaluation of the volume management.

More details about the algorithm options are described in the next section.

7.1.3 Algorithms and scenarios

The algorithms covered by the evaluation will leverage deterministic pathfinding. The algorithms against which SPSN will be compared are SABR implementations.

CGR (*cgr*) will be evaluated with the following options:

- Pathfinding options:
  - Minimize the hop count first (*-hop*).
  - The current element maintained during the shortest path construction is a contact instead of a vertex (*-cvic*).
- Alternative path construction options:
  - Use the limiting contact approach first ending (*-1st-end*).
  - Use the limiting contact approach first depleted (*-1st-dep*).
  - Use the adaptive yen algorithm (*-yen-ion*).
- Volume management options:
  - Use ETO (*-eto*).

  - Use ETO and Queue-delay (*-eto-qd*).
  - Contact segmentation (*-cs*).

SPSN (*spsn*) will be evaluated with the following options:

- Pathfinding options:
  - Single parent per vertex (*-basic*).
  - Minimize the hop count first (*-hop*).
  - Multipath tracking(*-mpt*).
- Interregional management options:
  - Enable shortcut support (*-shortcuts*).

All SPSN flavors use contact segmentation.

Regarding the pathfinding characteristics of CGR within aiodtnsim, the implementation leverages a node graph backend, and the contacts are easily accessible (nested maps) and sorted for each transmitter and receiver pair. The algorithm's core is way faster than a contact graph red-black tree design. Dijkstra's exploration stops as soon as the destination is reached. Work areas are also initialized, but on a node basis, the overhead is negligible compared to contact-based work areas. Consequently, pathfinding issues regarding the best route detection described in section 4.3 can occur.

When using option *-cvic*, the pathfinding approach falls back to an ION-like behavior by maintaining contact work areas and a current contact during exploration but still uses a node graph data structure. This confirms the statement made in section 4.1: contact graph shall refer to a data structure rather than a technique for maintenance of a current element. For CGR's second loop (election of the following current contact), a list of suitable candidates is maintained, rather than iterating over the whole contact plan [5].

The scenarios leveraged for evaluation are[6]:

- The *15s15g* scenario encompasses 30 nodes (15 satellites and 15 ground nodes).
- The *30s30g* scenario encompasses 60 nodes (30 satellites and 30 ground nodes).
- The *50s50g* scenario encompasses 100 nodes (50 satellites and 50 ground nodes).
- The *7r15s15g* scenario encompasses 7 regions, 15 satellites, and 15 ground nodes per region (210 nodes).
- The *127r5s5g* scenario encompasses 127 regions, 5 satellites, and 5 ground nodes per region (1270 nodes).

The scenarios *15s15g*, *30s30g*, and *50s50g* are realistic scenarios according to the predicted Martian network size by IOAG (figure 1.1). Scenario *7r15s15g* is the use case interregional scenario depicted in figure 2.1. In contrast, scenario *127r5s5g* proposes to scale up the regional tree size (but reduces the region sizes for simulation ease). The scenario sizes are designed to reach the network size requirements of hundreds of nodes and hundreds of regions. The interregional scenario reaching this requirement shows 127 regions due to the binary tree structure approach (depth of 6).

[5] A similar method can be observed in the unibo-cgr implementation (`https://gitlab.com/unibo-dtn/unibo-cgr`).

[6] Scenario names format similar to [FW17].

Listing 7.1: Regional configuration file example.

```
{
    "root_region": 1,
    "neighboring": {
        "1": [2, 3],
        "2": [4, 5, 1],
        "4": [2],
        "5": [2],
        "3": [6, 7, 1],
        "6": [3],
        "7": [3]
    },
    "shortcuts": [
        [5, 6], [7, 5], [5, 7], [6, 5]
    ],
    "membership": {
        "1": [
            "NANOSATC-BR1 (r1)",
            "PROMETHEUS 2-1 (r1)",
            "ZACUBE-1 (TSHEPISOSAT) (r1)",
            "gs0 (r1)",
            "gs1 (r1)",
            "gs2 (r1)"
        ],
        "2": [
            "AEROCUBE 4.5A (r2)",
            "BRITE-PL2 (HEWELIUSZ) (r2)",
            "CUBESAT XI-IV (CO-57) (r2)",
            "gs0 (r2)",
            "gs1 (r2)",
            "gs2 (r2)"
        ],
        "4": [
            "BRITE-CA1 (TORONTO) (r4)",
            "MOVE-II (r4)",
            "SIRION PATHFINDER-2 (r4)",
            "gs0 (r4)",
            "gs1 (r4)",
            "gs2 (r4)"
        ],
        "5": [
            "BEESAT-1 (r5)",
            "POLYITAN-1 (r5)",
            "SWISSCUBE (r5)",
            "gs0 (r5)",
            "gs1 (r5)",
            "gs2 (r5)"
        ],
        "3": [
            "AEROCUBE 8C (r3)",
            "DELFI-N3XT (r3)",
            "QB50P2 (r3)",
            "gs0 (r3)",
            "gs1 (r3)",
            "gs2 (r3)"
        ],
        "6": [
            "AEROCUBE 7 (r6)",
            "CORVUS BC4 (r6)",
            "FUNCUBE-1 (AO-73) (r6)",
            "gs0 (r6)",
            "gs1 (r6)",
            "gs2 (r6)"
        ],
```

```
        "7": [
            "AEROCUBE 6B (r7)",
            "BRITE-PL2 (HEWELIUSZ) (r7)",
            "CUTE-1 (CO-55) (r7)",
            "gs0 (r7)",
            "gs1 (r7)",
            "gs2 (r7)"
        ]
    }
}
```

Scenario name	Contact count	Contact volumes (bits)	Contact durations (seconds)	Scenario duration (hours)	Data rate (bit/s)	Ring road inter-satellite links
15s15g	3608	~4244001	~442	48	9600	Yes
30s30g	22242	~4295805	~447	48	9600	Yes
50s50g	63678	~4351558	~453	48	9600	Yes
7r15s15g	21144	~4353896	~454	24	9600	No
127r5s5g	160772	~4366237	~455	24	9600	No

Table 7.1: General metrics for each scenario.

Scenario name	Border nodes per region	Shortcuts	Border nodes per shortcut
7r15s15g	5	2	5
127r5s5g	5	10	5

Table 7.2: Interregional metrics for each scenario.

The scenario details are gathered in tables 7.1 and 7.2. Border nodes refer to nodes being receivers or transmitters of contact passageway.

For the interregional scenarios, each region is a ring road network, and the contact passageways are inter-satellite links. For simulation convenience, the regional membership of the nodes is known by all nodes from the beginning of the simulation, supplied alongside the regional neighboring leveraged to initialize the regional graph. The shortcuts are also provided alongside the tree neighboring but separately to enable or disable support easily for evaluation (an example is available in listing 7.1).

The values for contact data rates are the default values the framework provides. As this thesis focuses on routing rather than contact prediction, variations of the bundle sizes rather than variations of the contact data rates will be preferred.

7.2 Offline analysis

An offline method for isolated features is required for a fine-grained processing evaluation. Feature isolation renders online evaluation complicated (simulation against scenarios to retrieve network performance results). The *-basic* SPSN flavor is provided for convenience to compare with a standard Dijkstra algorithm.

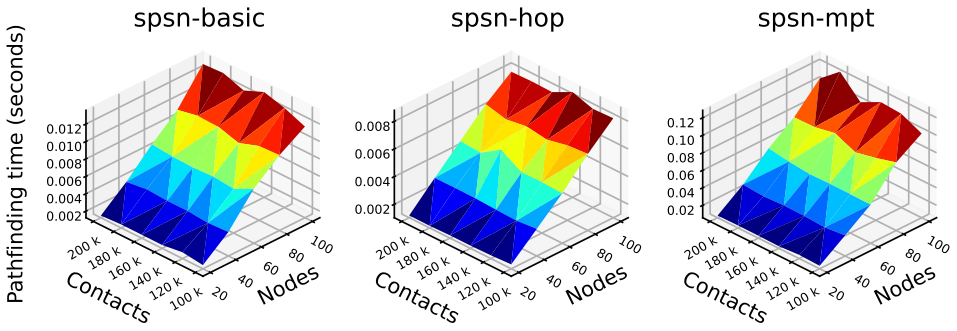


Figure 7.2: Tree construction times.

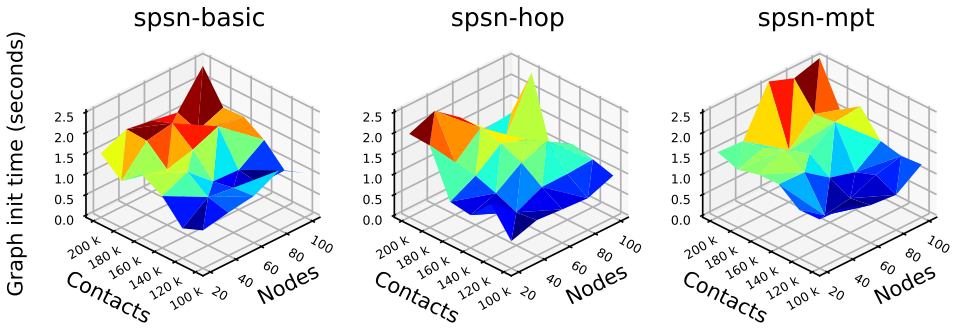


Figure 7.3: Graph initialization times associated with the simulations of figure 7.2.

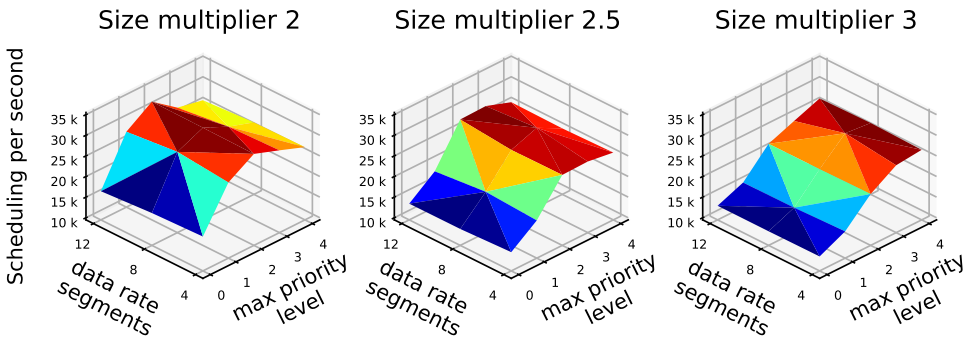


Figure 7.4: Scheduling per second for the enhanced manager.

The most important information is that shortest-path tree construction is successfully unaffected by variations of the contact plan horizon for the same node count, whatever tree

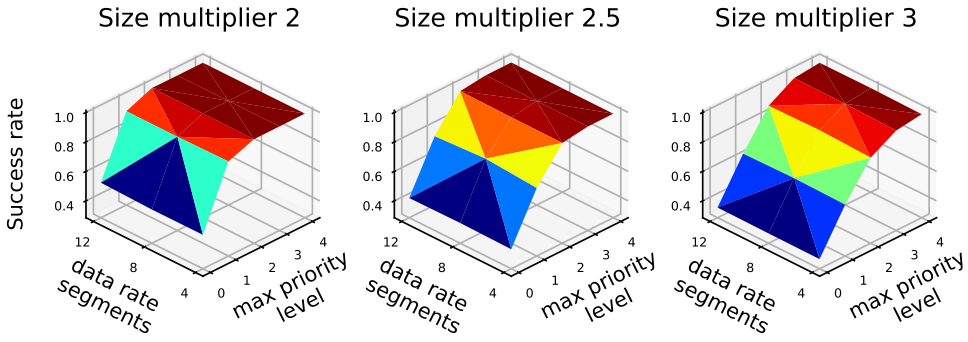


Figure 7.5: Scheduling success rates associated with the simulations of figure 7.4.

Contact duration (seconds)	120
Tx data rate (bit/second)	10000
Contact volume (bits)	100000000
Bundle count	10000
Average bundle size	10000 × multiplier

Table 7.3: Contact and bundle characteristics for figures 7.4 and 7.5.

construction technique is being leveraged. The processing time shall be very stable while the number of contact increases. In other words, SPSN shall be predictable, and as long as at most one tree is computed (by design), a predictability check of the tree construction shall be conducted.

Figure 7.2 depicts the offline evaluation results for the tree construction, and figure 7.3 provides the corresponding graph initialization processing times. Each three-dimensional point (contact count, node count, processing time) represents the average processing time over 20 tests for the same contact and node count. The contact plans used for evaluation are SABR compliant (no contact overlapping between two nodes) and randomly generated. The maximum egress link is limited to 20 (a node has contacts with at most 20 other nodes). Still, 20 ingress and 20 egress links can be active simultaneously. This will highly impact the pathfinding of multipath tracking while being hardly realistic, which does not limit the quality of the evaluation. As the main goal of this section is to show the predictability of SPSN, any network size sufficiently large fits this purpose.

The predictability check is successful, as when given a node count, the construction time remains stable while the number of contact (i.e. the contact plan horizon) increases.

On the other hand, the increase in the contact plan size shows a significant impact on the graph initialization by showing a volatile behavior. A better trend could maybe be observed if the number of tests was increased, but the main pieces of information that should be inferred from those results are:

- Contacts are generated randomly and appended to a list (contact plan). The ordering of

those contacts can affect the graph initialization due to its nested hashmaps structure.

- Contacts ordering for contact plan could be standardized to optimize initialization times.
- Contact plan updates remain rare. Initialization times of two seconds can be considered negligible for 200000 contacts (in comparison, the scenario *30s30g* shows only 22242 contacts for a two days contact plan horizon).

The graph initialization times are provided for informational purposes. The nature of the DTN topology representations, being multigraphs, offers contact plan ordering optimization, as highlighted by figure 7.3.

Contact segmentation in its enhanced version motivates fine-grained processing evaluation, as it maintains aggregates, segments, and bundle reference lists with relatively heavy list updates if the bundles are small, plentiful, and of highly variable priorities. Enhanced manager usage is still possible if priority and data rate variation support is not required (all bundles have priority 0, use of a single data rate segment). Still, the simple manager, evaluated in the next section, would be preferred.

The goal is to analyze the capabilities in worst-case scenarios. The evaluation metrics are gathered in table 7.3, and the results in figure 7.4, giving the bundle per second scheduling rates (one dry run and a volume update in case of dry run success), as well as the associated scheduling success rates in figure 7.5. One volume multiplier per column is considered in both figures.

The first aspect is the negligible impact of the data rate segment count. Indeed, the segments are considered only during the dry run phase, and the bundles are, in this scenario, not likely to overlap more than 2 segments. The overhead would also be negligible if the bundle would overlap more segments.

The second aspect is that the scheduling per second and success rates are lower if the maximum priority is low. This is coherent: if there is a reduced number of priorities, a bundle is unlikely to be able to reclaim an interval already used by another lower-priority bundle, reducing the success rate. Simultaneously, this inability to reclaim an interval requires more computational effort to search for an interval that can be claimed.

The third aspect is the decrease of the scheduling per second rate if the number of priority levels is superior to 3 (max priority being 2). This is imputable to a higher number of priority aggregates. If the max priority is relatively low, aggregate splitting and merging seem optimized, while a higher max priority could induce interval claim of various priority aggregates, complexifying the list management. Indeed, if the max priority is low, aggregates are expected to be longer, and booking would most likely involve a reduced count of aggregates.

SABR mentions three levels of priority. It is highly appreciable to opportunistically observe the best performance for the same amount of priority levels for the enhanced segmentation manager.

7.3 Eliminatory processing pressures

The reference implementation of CGR in ION maintains a current contact during exploration and uses Yen's algorithm for alternative route search construction, as suggested by the SABR standard. Therefore, this section evaluates Yen's algorithm (alternative route construction option) and the maintenance of a current contact during exploration (pathfinding option). Those options' expected processing times motivate a separate preliminary evaluation.

Scenario	15s15g	30s30g
Interval between injections (seconds)	80	20
Minimum bundle size (bits)	1600000	1600000
Maximum bundle size (bits)	1600000	1600000
Bundle TTL (seconds)	86400	86400
Last injection (hours after scenario start time)	32	32
Bundles injected	1445	5784

Table 7.4: Contact injection plan creation parameters for each scenario (20 different plans per scenario).

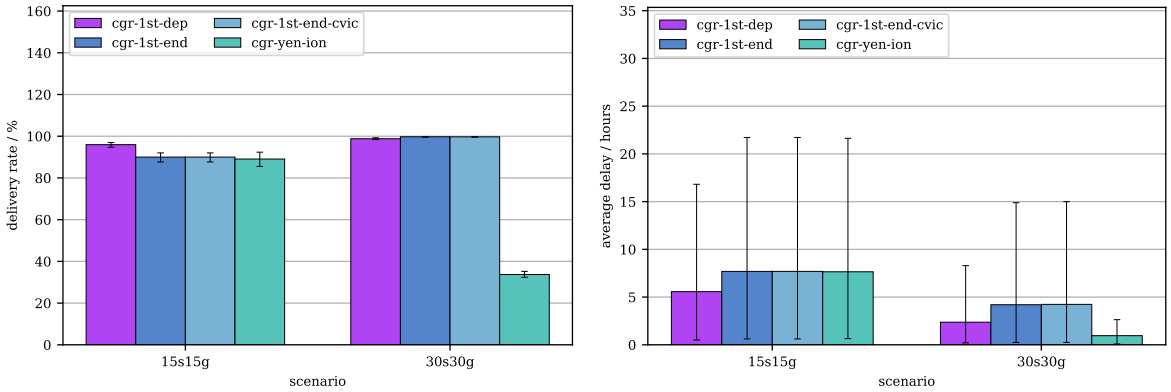


Figure 7.6: Delivery rates and delays for evaluating Yen's algorithm.

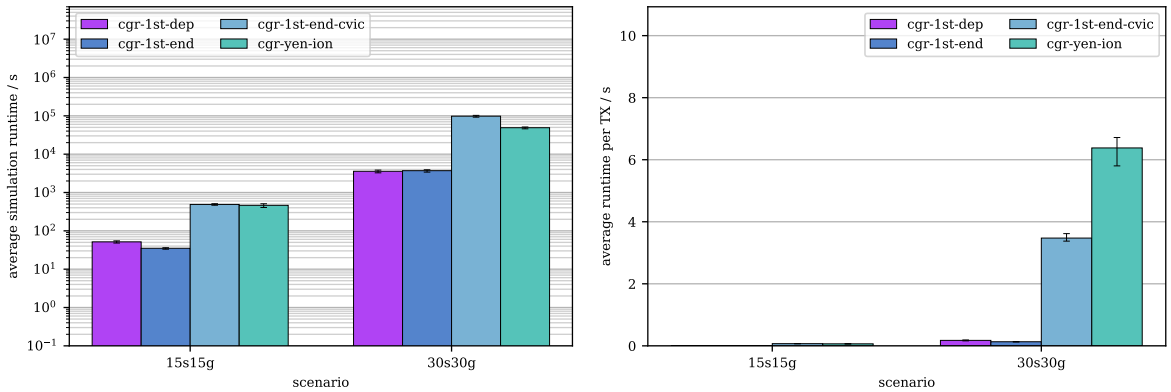


Figure 7.7: Simulation runtime and simulation runtime per transmission for Yen's evaluation.

They are therefore evaluated with relatively small scenarios (*15s15g* and *30s30g*), using the bundle injection plan creation parameters from table 7.4.

Moreover, to limit the processing time, the maintenance of the current contact will be tested with the first ending limiting contact approach for alternative route construction rather than Yen's algorithm (*cgr-1st-end-cvic*), while Yen's algorithm (*cgr-yen-ion*) will be tested with

the maintenance of a current node in addition to the following countermeasures:

- For a given bundle, CGR can only increment K for two iterations if Yen's main loop has to be resumed. If the newly found routes are still unsuitable, the bundle is dropped. This permits the mitigation of the computational hangs.
- The upper bound value of K is 1000. If K has already reached this value and the available routes are unsuitable for the next bundles (i.e., new routes have to be computed), the bundles are dropped.

Even if networking performance results are provided, the main focus of this section is the processing time. Results for other CGR flavors are provided as scientific controls (*cgr-1st-dep* and *cgr-1st-end*).

Using Yen's algorithm, those countermeasures do not permit the detection of a route for each bundle for the *30s30g* scenario, according to the figure 7.6. The superiority *cgr-1st-dep* over the other variants is imputable to congestion, as this variant would suppress the small volume contacts first. The inferiority of *cgr-yen-ion* for those metrics is imputable to the countermeasures.

Regarding the computational pressure, depicted in figure 7.7 shows that the countermeasures, while preventing the presence of hangs, do not, however, prevent the unsustainable increase of the computational pressure if the network size increases, with *cgr-yen-ion* presenting a 6 seconds average runtime per transmission for the *30s30g* scenario. The simulation overhead cannot cause this computational time, as the bundles are more likely to be dropped than transmitted. Completing a single test for this scenario already represents more than 13 hours of simulation time.

The scientific controls (*cgr-1st-dep* and *cgr-1st-end*) highlight that routes exist for the injected bundles. Consequently, most bundle drops of *cgr-yen-ion* can be considered unacceptable regarding the networking performance achievable with lower processing cost flavors. Addressing this issue, allowing CGR Yen's algorithm to perform better in delivery rate, would require lifting the countermeasures. As the processing time is already unrealistically sustainable by any resource-constrained spacecraft, a better-performing variant would also be unsuitable.

Using the current contact retention technique (*cgr-1st-end-cvic*), the networking performance is way better and also confirm the results found in [Wal20], at least for such heterogeneous topologies, stating that current vertex retention can show similar networking performance. The completion time of a single simulation is by far the worst, with more than 27 hours for the *30s30g*. However, the average runtime per transmission represents more than 3 seconds for this same scenario, which will still be unacceptable for the targeted network sizes. Last, the current state-of-the-art implementations, which combine the two approaches (current contact retention and Yen's algorithm), can also be considered unsuitable.

Further evaluation with comparison using Yen's algorithm or maintenance of a current contact during exploration is unrealistic in the context of a thesis about scalability. As already pointed out in [Wal20], the use of those variants is heavily impacted by the contact plan size, even with a node graph implementation of Dijkstra when using Yen's algorithm. The newly discovered issues in the preliminary analysis (section 4) further motivate an early exclusion of Yen's algorithm for the evaluation. Maintaining a current contact rather than a current node during exploration was also expected to show an increased processing time, and showing the limits of those approaches is sufficient to discard them (again, the current reference implementations of CGR use them conjointly).

This decision was first difficult from a theoretical point of view, as Yen's algorithm is supposed to be the state-of-the-art correct way to populate the routing table. However, the adaptive behavior, coupled with the possible issues of Dijkstra to find the best route when using the current node maintenance technique, weakens the argumentation that would justify the retention of Yen's algorithm for further evaluation.

From a practical point of view, the increasing sizes of the scenario rendered unrealistic the completion of the simulation within an acceptable timeframe due to the processing times of single simulations, the processing power required for possible parallelism, and the amount of memory needed. In any case, including *cgr-yen-ion* and *cgr-1st-end-cvic* for all scenarios represents simulation times measured in months. For the same reason, it can be observed that this preliminary evaluation of the *cgr-yen-ion* and *cgr-1st-end-cvic* flavors did not include the *50s50g* scenario. Testing this scenario would have been superfluous as the results on the *30s30g* scenario already showed discarding features.

7.4 Networking performance

7.4.1 Intraregional unicast routing tests

Scenario suffix	+lb	+mb	+sb
Interval between injections (seconds)	50	10	5
Minimum bundle size (bits)	1500000	30000	15000
Maximum bundle size (bits)	1700000	34000	17000
Bundle TTL (seconds)	86400	86400	86400
Last injection (hours after scenario start time)	32	32	32
Bundles injected	≈2300	≈11500	≈23000

Table 7.5: Contact injection plan for each scenario (20 different plans per scenario).

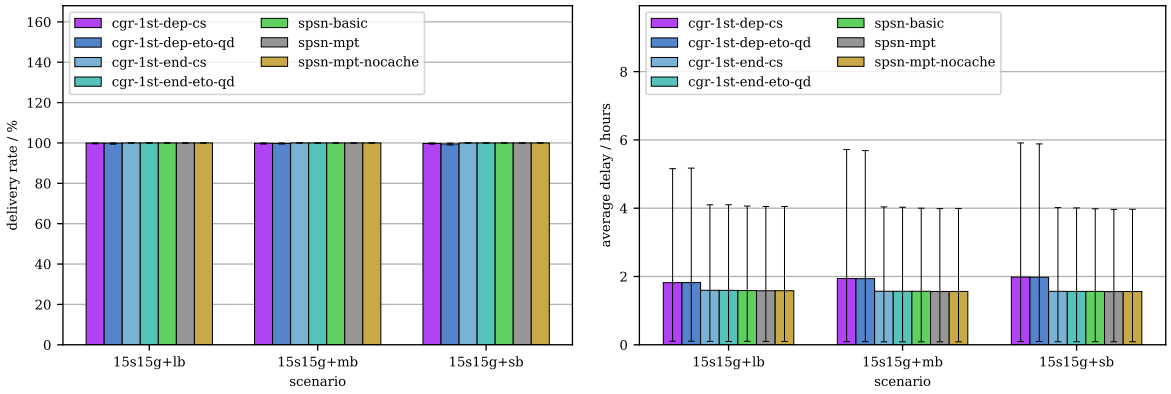


Figure 7.8: Intraregional delivery rates and delays (small topology).

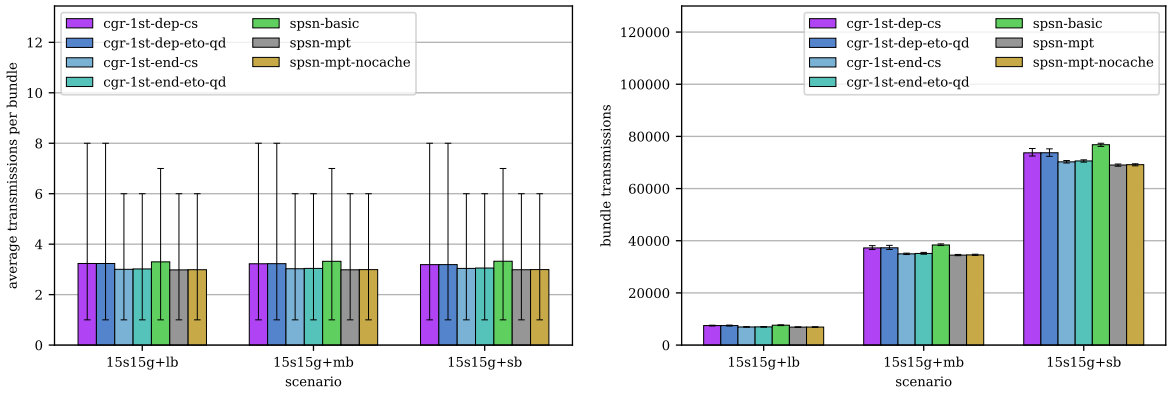


Figure 7.9: Intraregional hop and transmission counts (small topology).

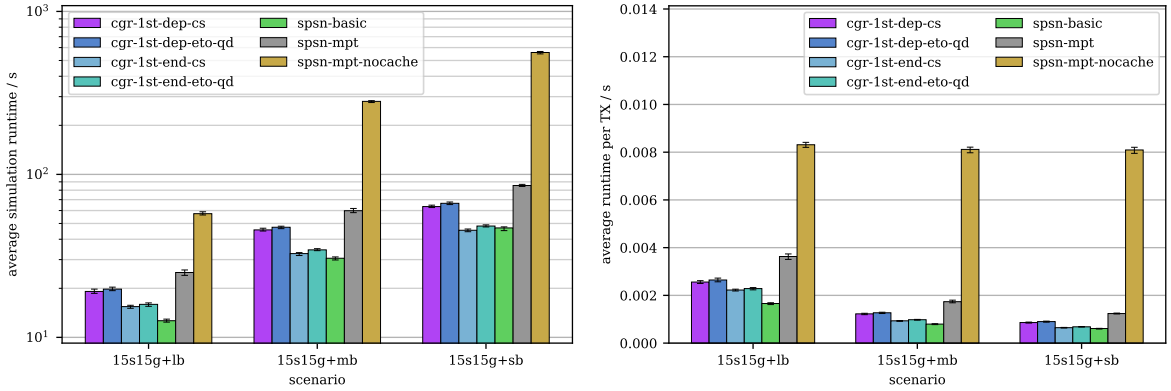


Figure 7.10: Intraregional simulation runtime and simulation runtime per transmission (small topology).

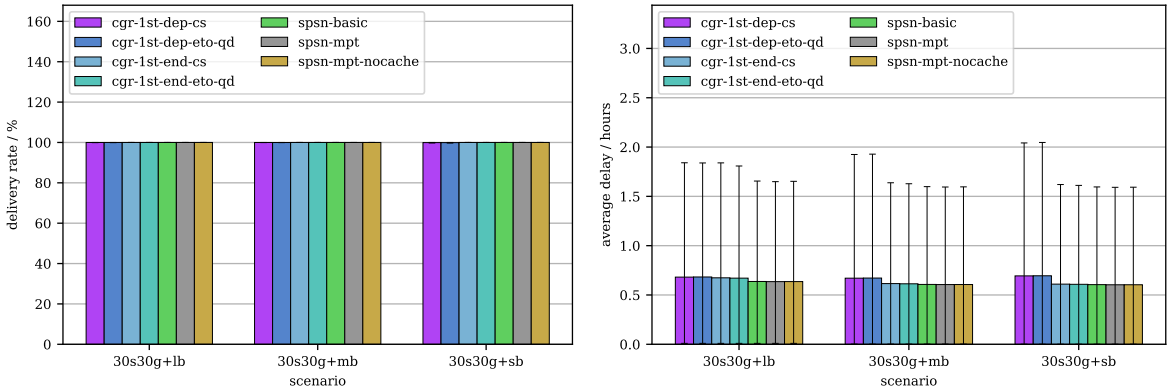


Figure 7.11: Intraregional delivery rates and delays (medium topology).

The evaluation of the unicast capabilities of SPSN is conducted thanks to three different injection plans, tested against the *15s15g*, *30s30g*, and *50s50g* scenarios.

Three sets of injection plan creation metrics will be used. Each set will be leveraged to create randomly 20 injection plans for each scenario, and for each scenario, each algorithm

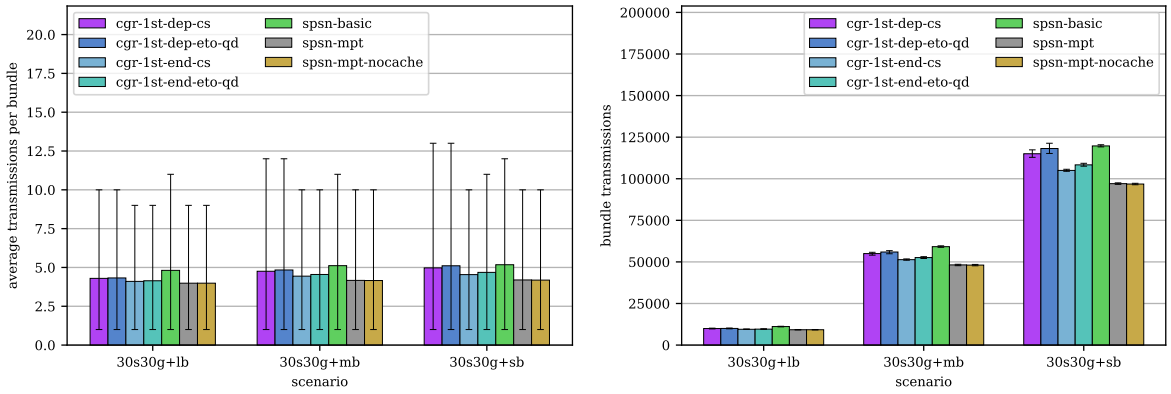


Figure 7.12: Intraregional hop and rescheduling counts (medium topology).

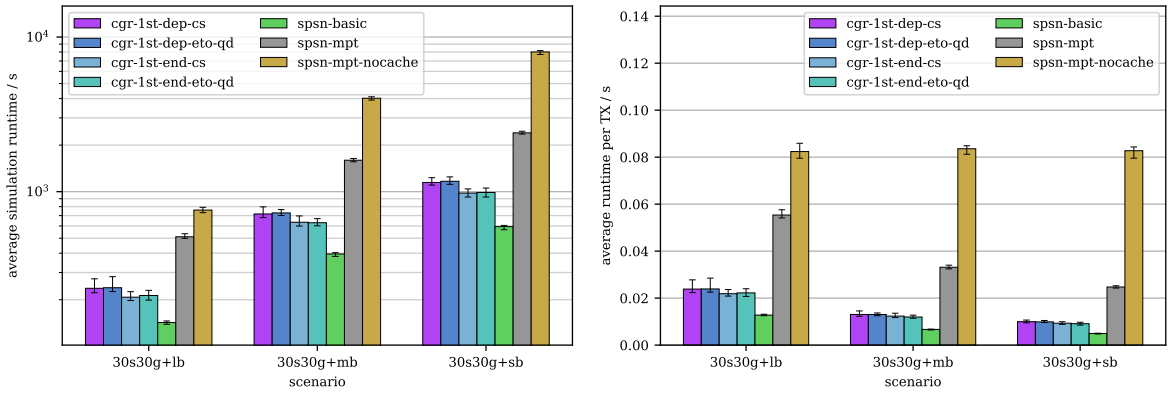


Figure 7.13: Intraregional simulation runtime and simulation runtime per transmission (medium topology).

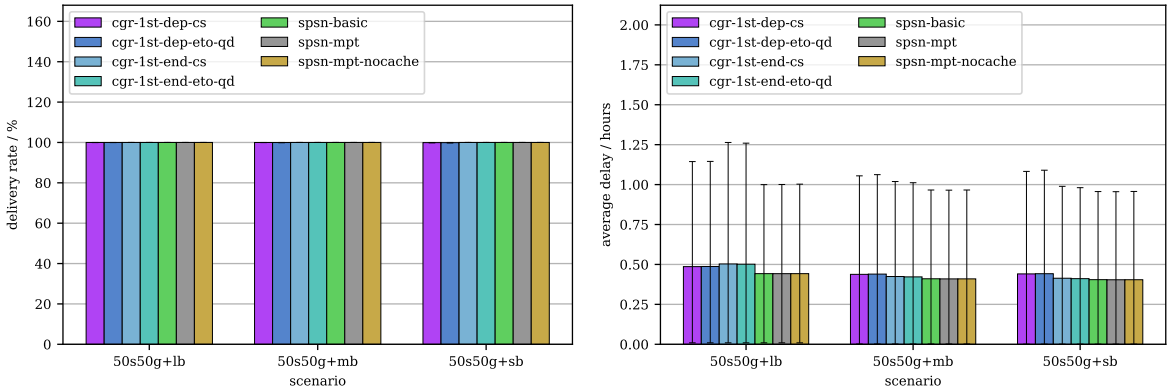


Figure 7.14: Intraregional delivery rates and delays (large topology).

will be tested with the 20 injection plans.

It shall be noted that if the metrics of a given set (e.g.+*lb*) are leveraged for injection plan creations for scenarios of different sizes, the load will be reduced for the larger scenario if compared with the smaller one. For example, the +*lb* set of metrics results in 2300 injections

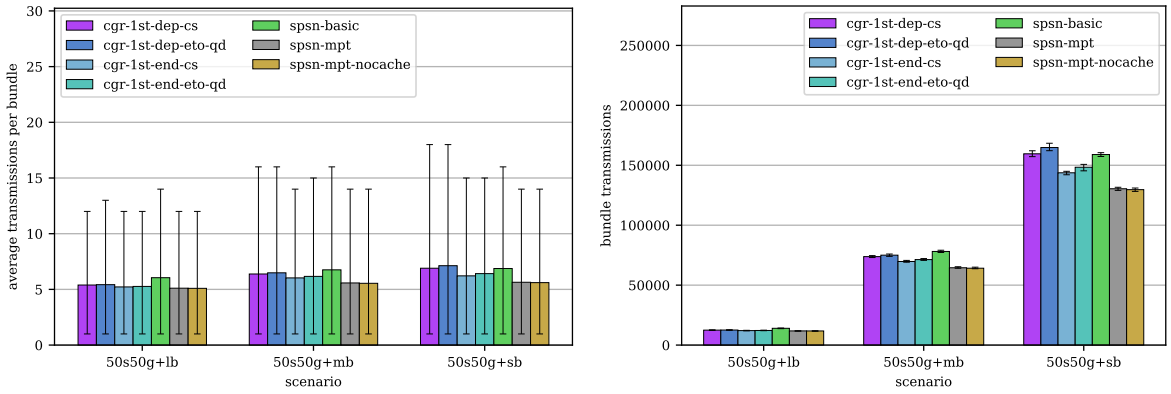


Figure 7.15: Intraregional hop and rescheduling counts (large topology).

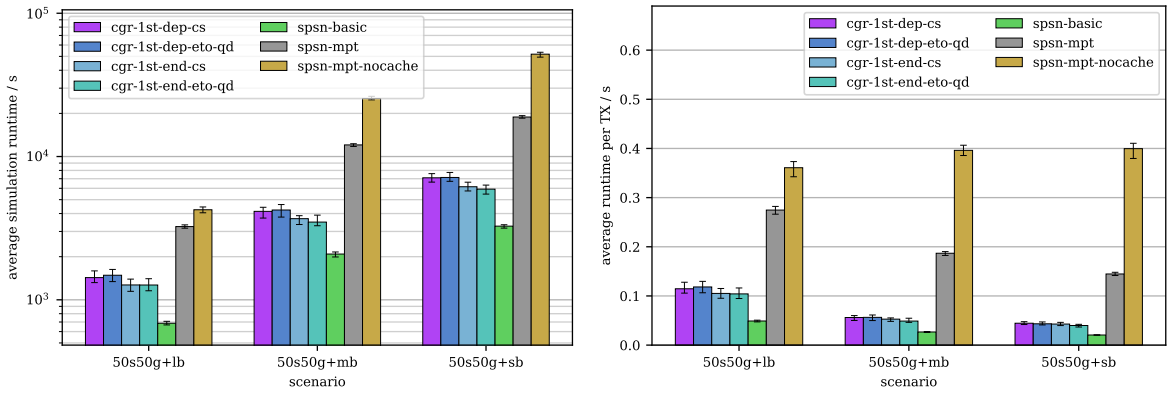


Figure 7.16: Intraregional simulation runtime and simulation runtime per transmission (large topology).

for both the *15s15g* and *50s50g*, and the former show fewer links to allow the bundles to reach their destinations if compared with the latter.

This aspect is not problematic, as space networks are scheduled, and operators would not create more load than the network can support. Any injection plan and scenario combination permits all bundles to reach their destinations. Table 7.5 gathers the injection plan creation metrics. The suffixes *lb*, *mb*, and *sb* stand for large, medium, and small bundles, respectively.

A larger scenario presents, however, a larger amount of existing routes to a destination. This higher amount of existing routes allows the simulation to highlight the pathfinding differences of the tested algorithms. If congestion is absent, the *-hop* variants of the algorithms can be discarded, and the SABR standard (optimizing the arrival time) is preferred.

On the *15s15g* scenario, the impact of SPSN can be considered limited. The delivery rates and delays do not present improvements (figure 7.8), and contact segmentation doesn't seem to impact the results for CGR (*-cs* variants). The CGR *-1st-dep* variants show higher delivery delays. The differences in average hop count and overall transmissions (figure 7.9) is also very limited. When the bundles are smaller but more numerous (*+sb* suffix), SPSN manages to lower by 1000 the total amount of transmissions (for a total of about 70000 transmissions, representing a decrease of about 1.7%). The difference can seem negligible but is noticeable compared to the other networking metrics considered.

Regarding the processing time (figure 7.10), multipath tracking shows a non-negligible overhead. It shall, however, be recalled that the reference CGR implementation SPSN was supposed to be compared with was CGR with Yen's algorithm, which was disqualified. Also, the results cannot be considered inadequate, as the average runtime per transmission does not exceed 4 milliseconds (simulation overhead included). The caching effectiveness, if compared with a zero caching strategy (*-nocache*), computing one tree per bundle, is twofold. Firstly, it can represent a decrease of up to 85% of the processing pressure. Secondly, reusing a tree does not seem to affect the networking performance compared with *-nocache*.

Recomputing a tree for each bundle ensures optimal performance. Reusing a tree can present suboptimal paths after a couple of bundle scheduling. The caching strategy and the recomputation of all paths (one per destination) if one of those paths becomes unsuitable, seems to be a valid heuristic for efficient networking performance and computation pressure tradeoffs.

By increasing the network size with the *30s30g* scenario, the impact of SPSN starts to be clearly noticeable. Considering the delivery delays (figure 7.11), the delays can be reduced by about 5% on average for the large bundle injection plans (*+lb*). This improvement can be considered very limited, but a clear trend starts to be observable when considering the average hop counts and total transmissions (figure 7.12).

If compared with the results of the *15s15g* scenario, the increase of the average path length (hop count) and the increase in the total amount of transmissions are imputable to the network size increase (even if an identical amount of bundles was injected). Longer hop count paths with earlier arrival times are indeed more likely to exist and sometimes undetectable by CGR. Conversely, CGR would sometimes leverage a route for scheduling even if a route with a lower hop count and the same arrival time exists.

Multipath tracking is designed to detect those better paths, and the results regarding delivery delays and hop counts confirm the expectations. On this *30s30g* scenario, a higher impact on the hop count can be observed, with more than a 7% decrease of the average hop count if compared with the best CGR variant, *cgr-1st-end-cs*, which represents a total transmission count reduction of about 8000 transmissions.

The positive impact of contact segmentation (*-cs*) can be observed with this network size. When using *-1st-end* alternative route strategy, CGR reduces the average path length by more than 3% by replacing ETO and queue-delay (*-eto-qd*) by contact segmentation. This reduces the total transmission count by about 3500. The improvement is limited but reflects the expected higher accuracy. Any potential gains might be considered significant regarding the financial costs of an extreme space environment: contact volumes are precious resources.

The processing times (figure 7.13) for SPSN are still a bit higher, but the same comments expressed for the results of the *15s15g* scenario are applicable.

Reaching network sizes of 100 nodes confirms the trend, with a decrease of the end-to-end delays of about 9% for the *+lb* injection plans (figure 7.14) when comparing SPSN with *cgr-1st-end-cs*. In the same way, a decrease of about 9% can also be observed for the average end-to-end path length (figure 7.15).

The total bundle transmission count makes this gap more visible while highlighting a decrease in the required transmission when contact segmentation is leveraged (about 3% for the *+sb* injection plans).

The processing time per transmission (figure 7.16) is again higher for SPSN compared to the limiting contact flavors. However, the pathfinding quality is enhanced, and section 7.4.4 will highlight that the limiting contact flavors decrease the processing time thanks to hardly acceptable tradeoffs. For the sake of fairness, the implementation of CGR also leverages

node multigraph implementation with a priority queue, while production implementations of CGR did not proceed yet to this transition. Still, the reference implementations use Yen's algorithm, and section 7.3 proved that node multigraph implementation does not permit CGR to show acceptable processing time.

7.4.2 Intraregional multicast tests

Scenario	15s15g+lb	15s15g+sb	30s30g+lb	30s30g+sb
Interval between injections (seconds)	500	50	50	5
Minimum bundle size (bits)	1500000	150000	150000	15000
Maximum bundle size (bits)	1700000	170000	170000	17000
Bundle TTL (seconds)	86400	86400	86400	86400
Last injection (hours after scenario start time)	32	32	32	32
Bundles injected	232	2312	2314	23136
Multicast members	5	5	5	5

Table 7.6: Contact injection plan for each scenario (20 different plans per scenario).

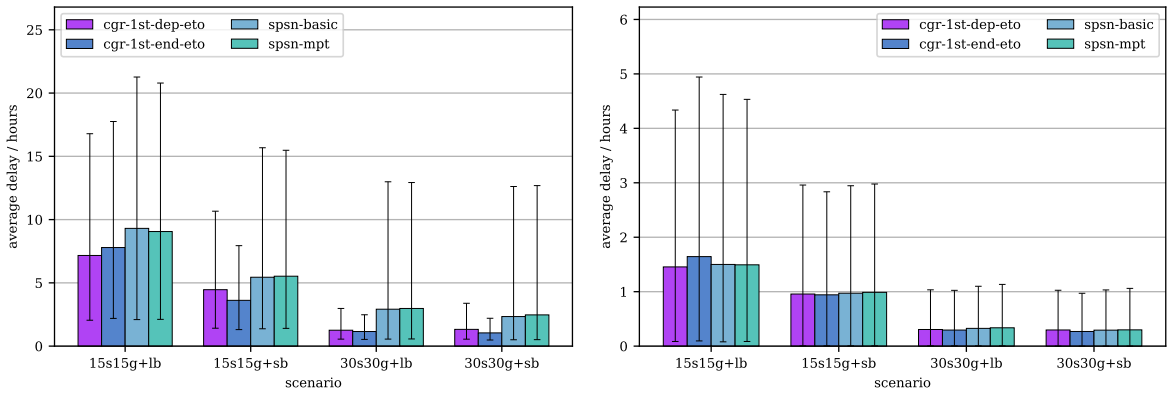


Figure 7.17: Multicast delivery delays (all copies on the left, first copy on the right).

Multicast support evaluation is based on the *15s15g* and *30s30g* scenarios, with two different injection plan strategies for each topology (table 7.6). Considering the figure 7.17, the results of SPSN can seem suboptimal regarding delivery delays.

However, the delivery rates (figure 7.18) show that SPSN is significantly more likely to deliver to all the group members. An exception can be observed for the *15s15g* with large bundles (+*lb*). However, the contact utilization on this topology is much higher (figure 7.20), and the results of the limiting contact CGR variants in such more congested scenarios can easily be misinterpreted, justifying the need for a dedicated discussion (section 7.4.4).

The most significant increase in the delivery rate is observed in the *30s30g+sb* scenario, with an increase of about 14% (from 82% to 93.5%). This can highlight that SPSN puts more effort into permitting each copy to reach its destination(s) and that CGR fails to find correct

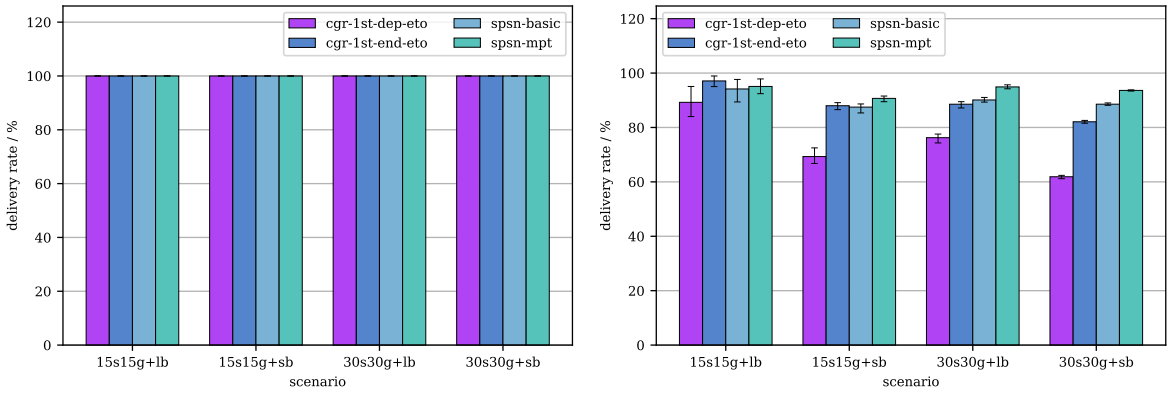


Figure 7.18: Multicast delivery rates (one member on the left, all members on the right).

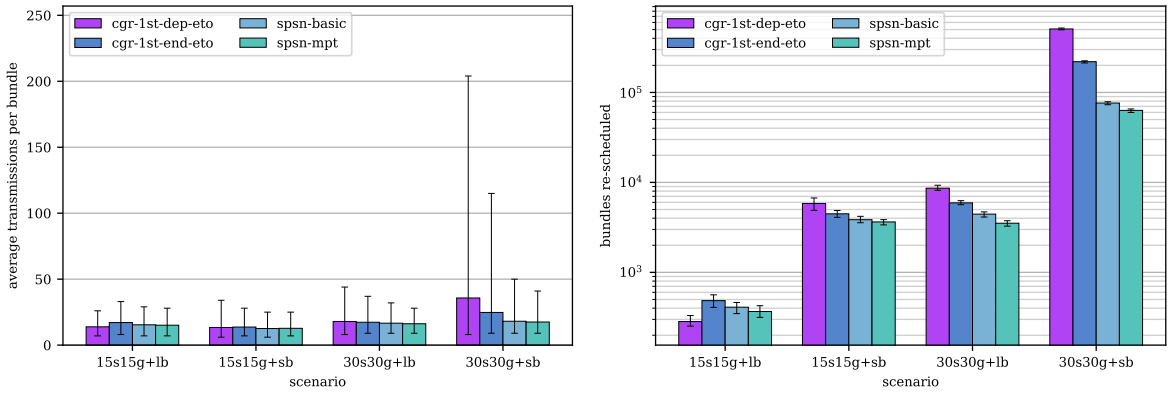


Figure 7.19: Multicast hop and rescheduling counts.

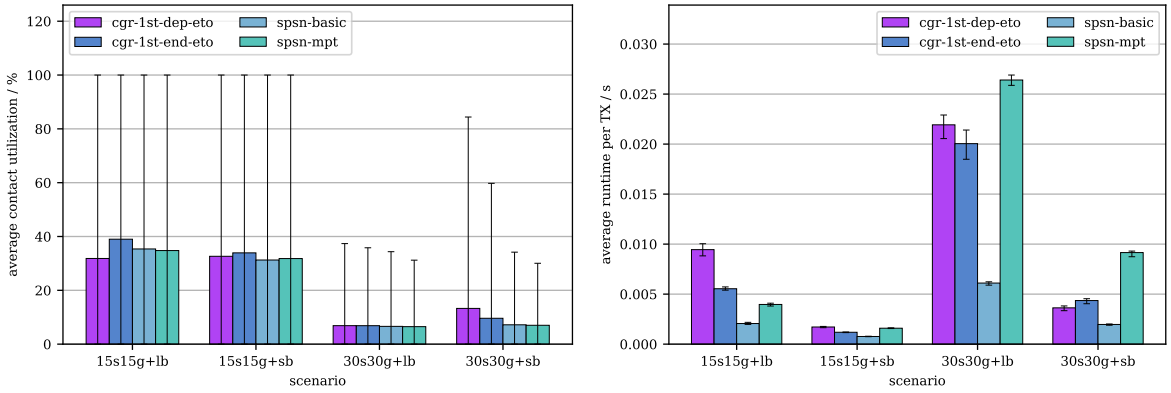


Figure 7.20: Multicast contact utilization and simulation runtime per transmission.

paths. Indeed, figure 7.19 shows that the average hop count is highly reduced with SPSN in this scenario, while CGR appears to reschedule a high amount of copies. A rescheduling means that the receiving node already knows the bundle (i.e., the presence of network loops).

It shall be noted that the network loops control is disabled for both algorithms. Therefore, those results shall be taken with care. Networks loops are more likely to occur with

CGR-multicast, as its volume management does not update all the contacts of the routes selected. This is more concerning than disabling network loop control, as the former is a design issue while the other is a configuration issue. In any case, this section compares algorithms developed by the same author. Handling of network loops will be further discussed in section 7.5.

The computational pressure of SPSN for multicast (with multipath tracking) is still a bit higher if compared with CGR (figure 7.20), but again, due to the unrealistic possibility of comparing SPSN with the actual reference implementation being *cgr-yen-ion*. Regardless of the other CGR variants, the average simulation runtime per transmission remains short, with more than 20 milliseconds in the worst cases (simulation overhead included).

7.4.3 Interregional routing tests

Scenario	7r15s15g	127r5s5g
Interval between injections (seconds)	10	100
Minimum bundle size (bits)	1500000	1500000
Maximum bundle size (bits)	1700000	1700000
Bundle TTL (seconds)	172800	172800
Last injection (hours after scenario start time)	16	16
Bundles injected	5777	1158

Table 7.7: Contact injection plan for each scenario (20 different plans per scenario).

The support of interregional shortcuts will be evaluated with a single injection plan set of metrics for each scenario, gathered in table 7.7.

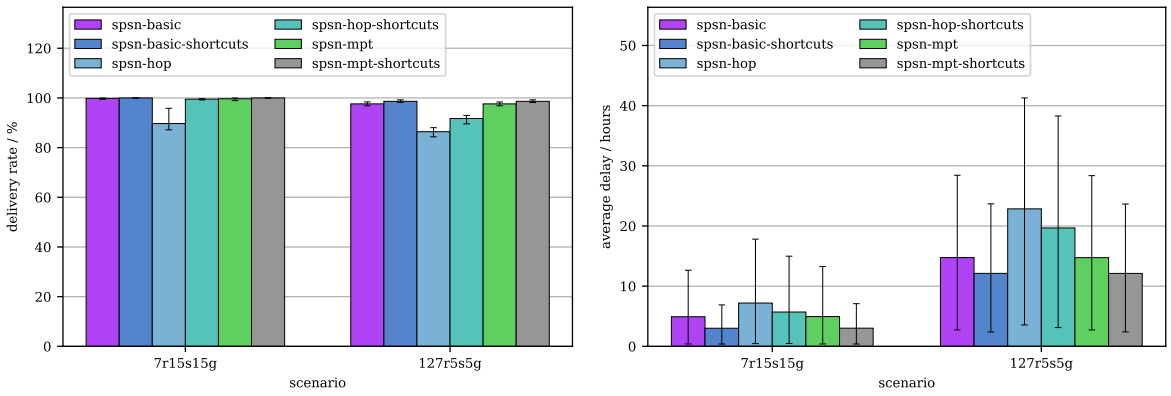


Figure 7.21: Interregional delivery rates and delays.

This evaluation intends to test the correct functioning of the contact passageway design, including the pathfinding and the interregional interfaces.

So the comparison with the approach depicted in 3.1.6 is hardly possible because still being subject to continuous non-negligible changes while operating with different assumptions and providing different features.

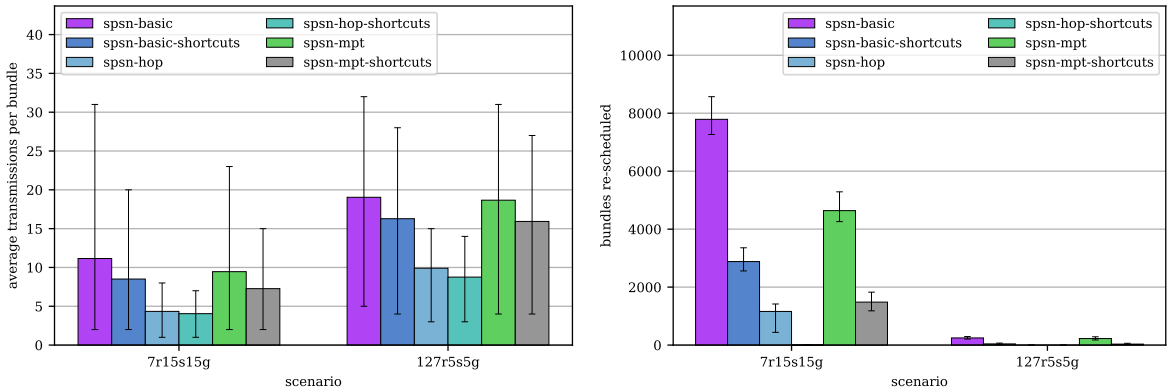


Figure 7.22: Interregional hop and rescheduling counts.

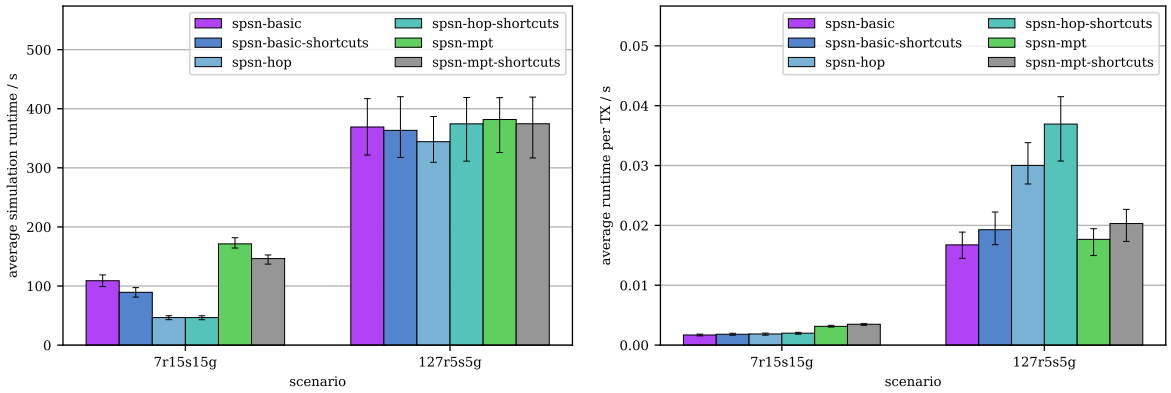


Figure 7.23: Interregional simulation runtime and simulation runtime per transmission.

The evaluation of the shortcut support impact can be considered more as a use case for validating the concept rather than a means to answer a real research question. Effective interregional pathfinding can be leveraged with shortcuts will, unsurprisingly, increase the throughput and decrease the delivery delays, as depicted in figure 7.21.

As expected, the average hop count decreases when shortcuts are present, and networking loops are less likely to occur if interregional throughput increases (figure 7.22).

Concerning the computational pressure (figure 7.23), shortcut support slightly increases the simulation run time. A slight increase could indeed be expected. When a shortcut is added in the scenario *127r5s5g*, two virtual nodes are added (one per region), which increases the node count by 50 as 10 shortcuts are added. Adding an extra region has a smaller impact than adding a node. The node translation (getting the next virtual node hop from a distant node id) uses a quasi-static forwarding table, while the node count impacts the tree construction. SPSN tree (re-)construction is more likely to happen in comparison with the interregional routing table update. Last but not least, the runtimes provided do not exclude the simulation runtime, and it could be observed that the average transmission count per bundle was significantly higher in the *127r5s5g* scenario.

Scenario name	Contact count	Contact volumes (bits)	Contact durations (secs)	Scenario duration (hours)	Data rate (bits)
fg1mb+hg	16138	~457000	~48	8	8000000

Table 7.8: General metrics for each scenario.

Scenario	15s15g	30s30g	fg1mb+hg
Interval between injections (seconds)	80	10	1
Min bundle size (bits)	1500000	1500000	44000000
Max bundle size (bits)	1700000	1700000	46000000
Bundle TTL (secs)	86400	86400	24000
Last injection (hours after scenario start time)	32	32	8
Bundles injected	1445	11568	28740

Table 7.9: Contact injection plan for each scenario (20 different plans per scenario).

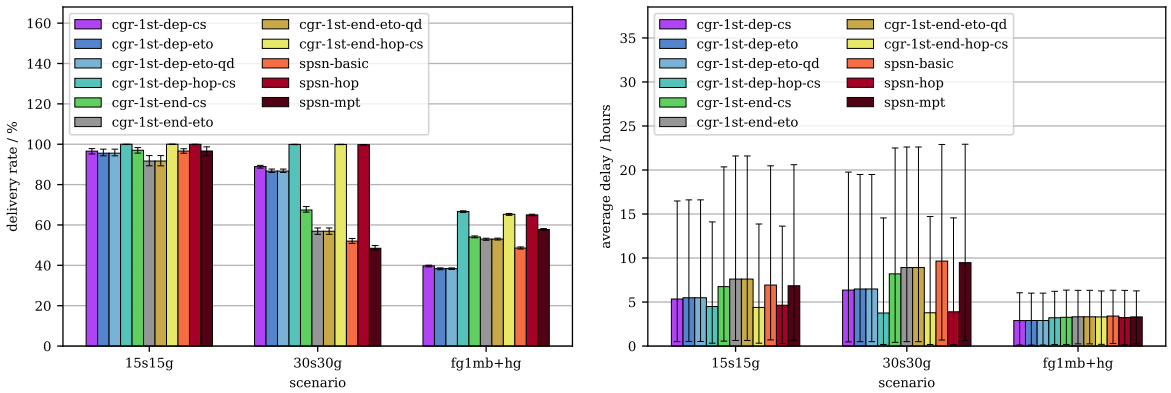


Figure 7.24: Delivery rates and delays.

7.4.4 Behavior with congestion

This section will cover CGR behaviors with the support of congested scenarios. To this end, the additional *fg1mb+hg* topology (206 nodes) is introduced (table 7.8). This public transportation scenario of the city of Freiburg is particularly interesting for testing as it encompasses very heterogeneous node mobility patterns. The weakness of the previously considered scenario is their relative homogeneity. Most nodes, satellites or ground stations, are reachable via multiple paths. This factor is increased by late contact plan horizon, while in a transportation scenario, some nodes might be reachable only via vehicles driving through specific roads. The introduction of this topology can be motivated by potential future Mars surface networks encompassing line mobility patterns (e.g., public trains showing contacts with stations or mining transportation nodes).

The limiting contact variants of CGR might suppress important contacts, with more serious consequences when compared with more homogeneous networks. Analysis of that matter

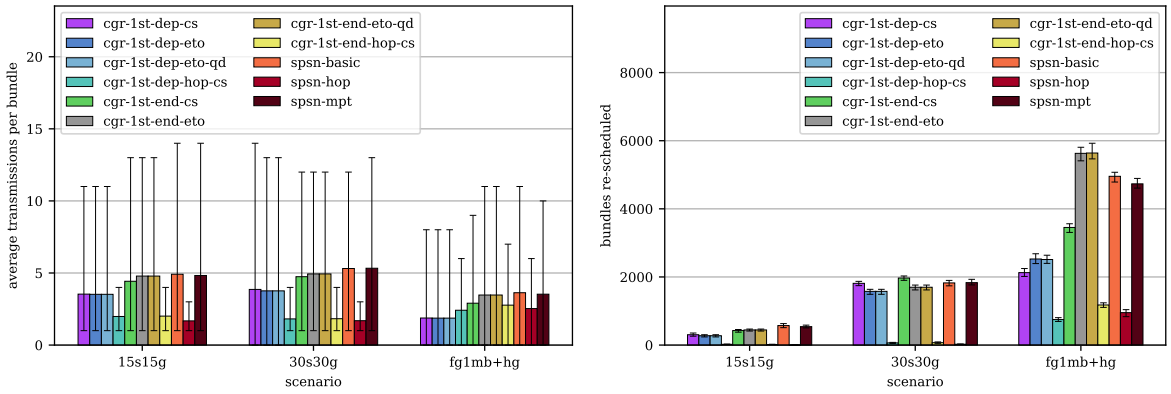


Figure 7.25: Hop and rescheduling counts.

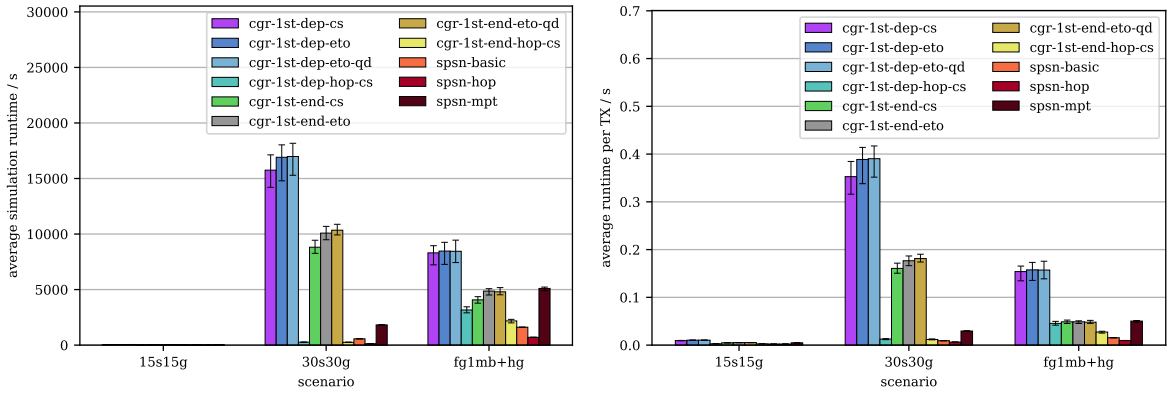


Figure 7.26: Simulation runtime and simulation runtime per transmission.

is essential, as future space networks can be much more complex and encompass node mobility patterns or topological characteristics that would render the suppression of some contacts dramatic.

Table 7.9 gathers the injection plan creation metrics.

The main purpose of this section is to recall that, even if SPSN was mainly evaluated with comparisons against the limiting contact approaches, those variants can sometimes present unacceptable behaviors, which are at best undetectable by basic network performance evaluation, and at worst even increase those performance. A lack of awareness of those behaviors could lead to misinterpretations about the real behavior of those variants.

In a homogeneous topology, the suppression of contacts shows a rather limited impact if congestion isn't present, as multiple paths are available. But if congestion occurs, suppression of contacts might casually render some routes undetectable.

At first glance, this is a drawback for network performance. But congestion renders this behavior advantageous on a macro scale for homogeneous networks, as limited premature drops increase the probability of the other bundles reaching their destination.

Analysis of this behavior proved that bundles were casually dropped very early during simulation, sometimes directly at the source. In contrast, the same bundles have been scheduled when using SPSN (tested while discarding possible load perturbations).

From a micro scale, e.g., a local node point of view, this is unacceptable to drop a bundle if

a suitable route exists. This can be considered quite unfair behavior for evaluation, as it can increase the overall network performance even though the cause is an evident weakness.

For instance, taking the delivery rates into account (figure 7.24), it can be observed that this behavior permits the non *-hop* CGR variants to perform well in homogeneous networks if congestion increases while the performance were similar or slightly lower than SPSN's before congestion (scenarios *15s15g* and *30s30g*). Congestion can be asserted as only the *-hop* variants (known to reduce the contact utilization) reach a delivery rate of 100% in the *30s30g* scenario, with a more than significant margin. In the same scenario, the delivery rate of SPSN completely dropped, with about 48%, against more than 56% for *cgr-1st-end-eto-qd* (an increase of 15%).

But switching to a heterogenous network like *fg1mb+hg*, the deception does not hold true, and the lower congestion induced by the premature drops does not counterbalance the impact of important contact suppressions anymore. SPSN reaches a delivery rate of more than 57% against about 54% for *cgr-1st-end-cs* (an increase of 6%). That behavior consequently allows a decrease in the hop counts, as shown in figure 7.25. The lower rescheduling count (network loops) for CGR variants that perform worse than SPSN in the *fg1mb+hg* can be due to premature drops. SPSN does not present this issue.

Regarding computational pressure, the *fg1mb+hg* scenario presents the most similar results between SPSN and CGR for a relatively short scenario duration (8 hours). Previous sections showed that SPSN is not impacted by late contact plan horizon when compared with CGR. The higher scalability of SPSN, if compared to Yen's algorithm variant of CGR, did not oppose any resistance. The results of the *fg1mb+hg* are therefore promising for the scalability of SPSN with later horizons, including comparisons with the limiting contact variants of CGR.

7.5 Requirement fulfillment

The evaluation proposed to test SPSN with the support of CGR variants for comparison. Unfortunately, direct comparison with the state-of-the-art variant of CGR proved to be hardly possible. This evaluation aspect can be considered a negligible issue, as Yen's algorithm could be discarded for scalability thanks to strong arguments. The inherent computational pressure of Yen's algorithm (its algorithm complexity), coupled with the presence of hidden variables (single scheduling can trigger hangs) and pathfinding inaccuracy (incapacity to find the best path in a single search, unable to leverage the best path due to the adaptive behavior) proved CGR with Yen's algorithm to be unrealistically applicable to large networks, with an apparent absence of SABR-compliance. Even if the route selection is effectively SABR-compliant, this can be considered only as a facade as long as the routing table populating phase cannot insert in time the best existing route for a bundle to schedule.

The issues (mentioned above) of CGR with Yen's algorithm are shared by the limiting contact approaches, as they share the same behaviors (adaptive implementation, Dijkstra stops as soon as the destination is reached). They provide a relatively good reference for comparison. Those variants present, however, at best, unfair behavior and, at worst, unacceptable pathfinding strategy, as covered in the section 7.4.4.

Pathfinding

SPSN exhibited equivalent or better networking performance compared to those variants in realistic scenarios. In a congested environment, SPSN can still improve the performance in heterogeneous challenging topologies, while the apparent decrease in performance in homogeneous networks is imputable to the unfair CGR behaviors mentioned above. The requirement /R1/ is successfully fulfilled.

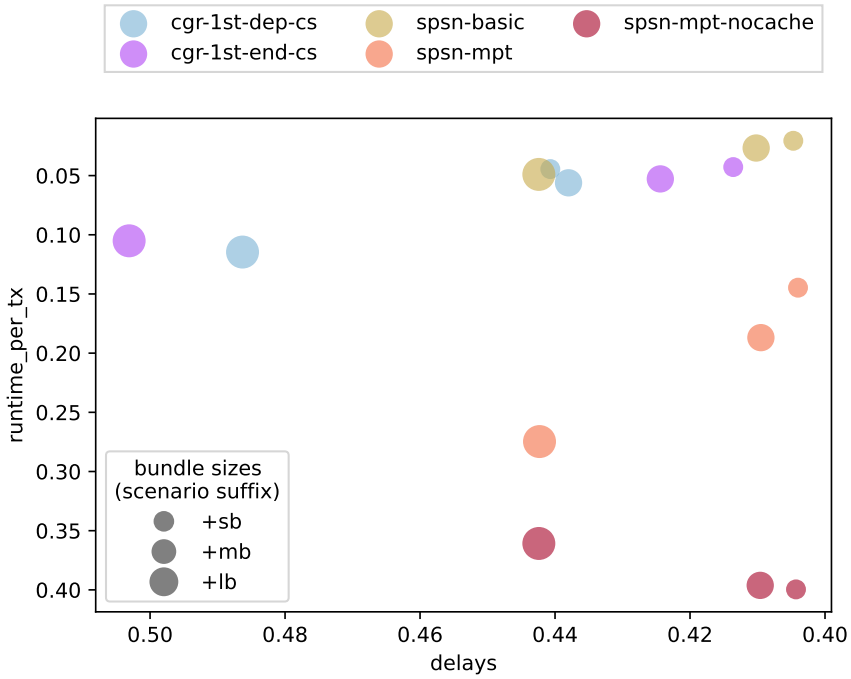


Figure 7.27: Intraregional evaluation summary for delays in the *50s50g* scenario.

The offline analysis of SPSN also proved the predictability of the computational pressure. With a given number of nodes, SPSN's tree construction time is entirely resilient to the contact plan horizon. Moreover, the guarantees that, at most, one tree will be computed for a bundle and that a tree will at least be used once for routing purposes renders SPSN's processing pressure highly predictable, discarding the risks of memory usage and processing time explosions, fulfilling requirement /R2/.

If the SPSN and CGR multicast integrations had to be compared, it could be pointed out that multicast was integrated around unicast in the case of CGR, while SPSN took the path of considering unicast as a special multicast case. Designing a more encompassing solution provides straight forward support for reduced-scope use cases while extending a solution to use cases they were not designed for can be challenging. The main components of SPSN proved to be effective for the multicast load, with more accurate volume management. However, disabling the network loop controls for multicast evaluation can arguably motivate only a partial fulfillment of requirement /R3/. Volume management conflict can occur when using CGR or SPSN with multipath tracking. Multicast deserves more time and attention and further investigations, as the options to address this aspect are plentiful: embedding in pathfinding countermeasures to the volume management challenges, creating one copy per exclusion list set, overbooking the ganging process by duplicating some destinations among several

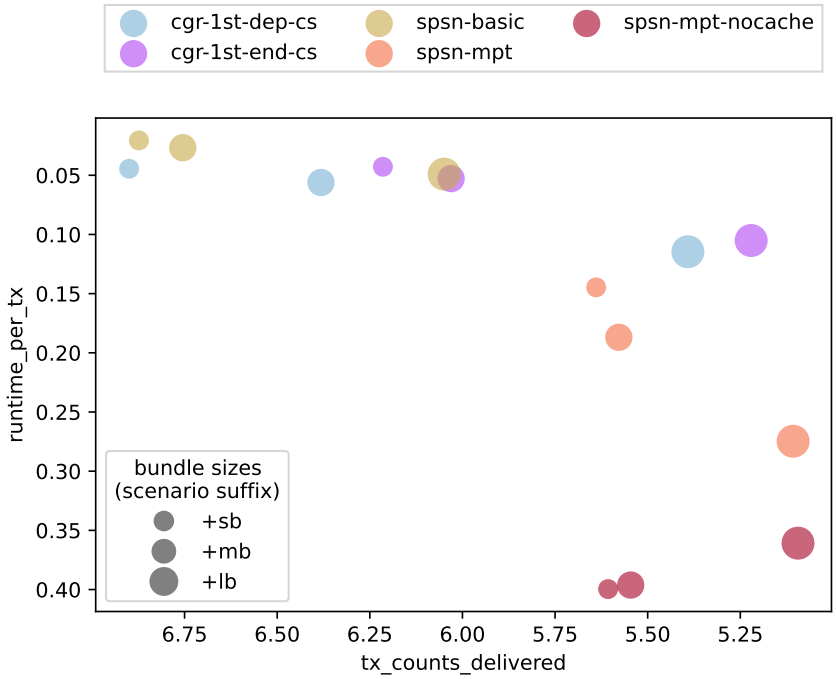


Figure 7.28: Intraregional evaluation summary for hop counts in the *50s50g* scenario.

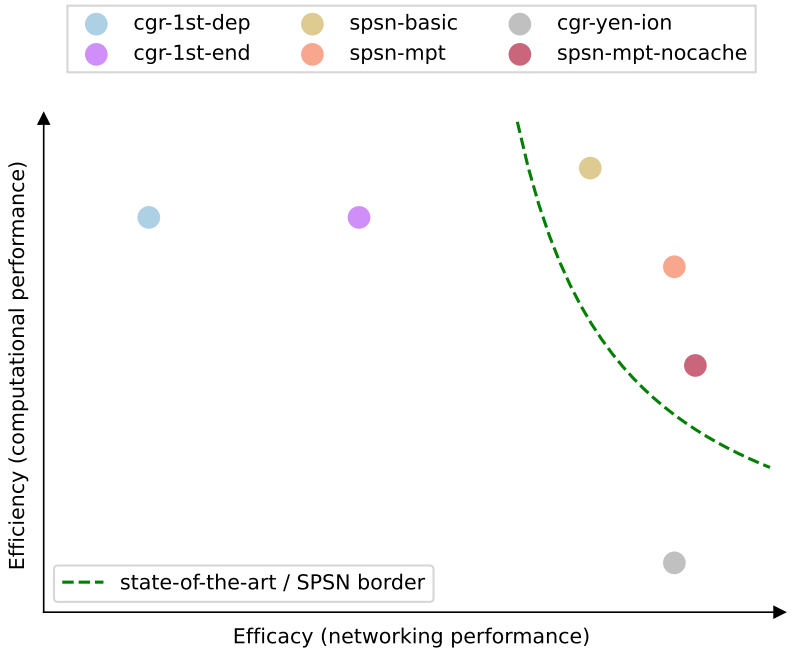


Figure 7.29: Summary of the trends observed during evaluation. Derived from figures 7.27 and 7.28, sections 7.3 and 4.

gangs, decrease volume management accuracy to decrease the copy count, just to name

a few aspects. This concern is shared with CGR, preventing CGR from fulfilling this requirement. A simple solution would be to use a basic tree construction. Basic construction will also reduce contact utilization if the multicast member is large.

Figures 7.27 and 7.28 are provided to represent the computational pressure and networking performance tradeoffs visually. The axes' directions are inverted to allow the target area to be located in the top right of the figure (the goal is to decrease the hop counts, the delays, and the average runtime per transmission). Regarding efficacy (delays and hop counts), the multipath tracking flavors are consistently better than CGR, when comparing dots of the same sizes (same injection plans). The basic tree construction allows a better efficacy for the delays but not consistently for the hop counts. For CGR, the first ending approach is usually better than the first depleted flavor. Regarding efficiency, the CGR flavors are similar but less efficient than SPSN with basic tree construction. Multipath tracking requires slightly more resources than CGR, but those CGR flavors exhibit unacceptable behaviors to achieve this efficiency. If no caching strategy is organized for SPSN, efficiency can be improved in some cases, but with a non-negligile processing pressure overhead.

To allow the presence, within a single summary figure, of the reference implementation (CGR using Yen's algorithm), the dots can be placed thanks to the networking performance trends for the *50s50g* scenario (figures 7.27 and 7.28), experimental observations (section 7.3), and theoretical expectations (section 4). Consequently, the dot for CGR using Yen's algorithm is placed to show similar efficacy but very low efficiency in figure 7.29.

This scatter plot is provided for convenience, and the placements should be taken carefully. For example, the current element retainment strategy being leveraged can impact the networking performance of CGR and consider the best possible performances (in line with multipath tracking with caching). However, the scale of the axes was not assumed to be linear during the placement, and the presence of *cgr-yen-ion* in this figure does not express in any case some degree of suitability.

Volume management

Concerning the requirement /R4/, contact segmentation provides a better volume awareness, with the condition to operate with an accurate contact plan. Contact segmentation tracks precisely the intervals of bandwidth utilization for the transmissions of a given bundle. The approach is resilient to the edge cases. The decrease of the hop count in complex topologies when CGR operates with contact segmentation supports this statement. Contact segmentation is expected to perform well if the topology is complex, with frequent subsequent contact overlapping along the paths. If the contact plan is not accurate, the introduction of the abstract volume management interface permits to use of heterogeneous techniques for the contacts of the network, i.e., contact segmentation if the prediction accuracy confidence is high and another technique otherwise. The fulfillment of the requirement can consequently be considered successful.

With data rate variation and priority support, the enhanced manager proved that those constraints are not incompatible with contact segmentation. This proof of concept fulfills requirement /R5/. The evaluation coverage of the enhanced manager was, however, limited. A computational pressure analysis is proposed in the offline analysis section, but no networking performance evaluation was included. Of course, the abovementioned volume conflicts also need to be addressed. Even if proof of concept simulation was conducted successfully, the priority handling could bring important questions: shall the priority support be

processed only for bundles with the same source? Can the transmission queues be sorted by priority? Shall priority support be observed only in the source home region? Is priority appropriate in a multi-tenant interplanetary network? Also, priority support is, in essence, schedule destructive and decreases the delivery rate. It can be considered hardly scalable if priority variations for the bundles are frequent.

The requirement /R6/ can hardly be evaluated with simulations and can be considered rather subjective. Some arguments can be provided to support at least its partial fulfillment. Firstly, the abstract interface provides high flexibility for future development in volume management, as covered in the concept description chapter. Secondly, considering contact segmentation, it can be observed that segmentation can authorize linear data rate variation support because it provides accurate bandwidth utilization tracking. The real impact of the following proposal is speculative: contact segmentation could allow data rate variation support by discarding the mean data rate segments, replacing them with a single segment encoding the data rate variation as a function of time. For a given bundle size and transmission start time, the correct transmission end time, respecting linearly, for example, the characteristic bell shape data rate over the time curve of a ground-to-LEO-satellite transmission, could be calculated to update the bandwidth utilization intervals accurately. Polynomial regression could be leveraged to define those functions.

Interregional support

The contact passageway design permits the scheduling of bundles to the neighbor regions using the shortest path leading to any node of those regions by abstracting the non-home region membership with virtual nodes within the graph representation. The benefit is twofold: firstly, any network node operates with a single contact plan, and the graph size is increased by adding a single vertex per neighbor region, whatever number of nodes and contacts are involved in the interregional interface. Secondly, this renders the handling of interregional bundles within the home region identical to an intraregional bundle while optimizing the throughput and delays by leveraging the underlying intraregional routing capabilities in a load-balancing and high-availability fashion (thanks to volume management and multiple nodes acting passageway simultaneously). The requirement /R7/ is therefore fulfilled.

The approach is highly flexible, as a given node can be a border node with an arbitrarily high number of neighboring regions without overhead. The configuration is simplified. The contact plan supplied to a border node is identical to that provided to a non-border node. Forwarding using a next-region mechanism rather than a next-passageway mechanism at each interregional hop is also more flexible. The interregional forwarding table remains valid if some border nodes with a neighbor region lose their passageway behavior after a contact plan update (or upon node failure). The nodes would know the next regional hop for a given destination region. The forwarding would, however, require the source to know the home region of the destination node (to be attached with an extension block set as the destination or by encapsulating the bundle, as the other nodes along the path might miss such information). This could be seen as a weakness (due to the need for remote area knowledge), but it can be considered a strength. Firstly, with such handling, pathfinding relies on the knowledge of a single node (the source), while a next-passageway forwarding mechanism makes pathing rely on as many nodes as node passageways along the end-to-end path by actually diluting the information among all involved passageways. This is a rather high-risk choice, as this results in as many single points of failure as passageways (if they are unique) or in-

terregional interfaces along the path. Secondly, knowing the path to a region is sufficient to route any bundle having as destination a node that is a member of this region, decoupling interregional pathing and regional membership. The pathing is not passageway-based but region-based. The support of shortcuts can be seen as an excuse for discarding the hierarchical tree structure and leveraging more advanced interregional routing pathfinding. An argument for an interregional hierarchical structure can be the absence of possible network loops. This is a weak argument: the intraregional topologies are way more dynamic than the predicted quasi-static interregional structure, and still constraining a hierarchical tree structure inside a region to discard possible loops would be unacceptable. And again, the absence of multiple paths creates single points of failure. Free from those potential drawbacks, the contact passageways fulfill the requirement /R8/.

The need for a unique contact plan for any node of a region (including border nodes) lowers the operational tension, successfully fulfilling requirement /R9/. The computational overhead of the neighbor region nodes would be negligible as a single virtual node would be added to the neighboring region's contact plan.

8 Summary and Outlook

8.1 Conclusion

This thesis shows a strong emphasis on SABR, and the evaluation was centered on comparison with different CGR implementation options. This decision was motivated by the apparent conceptual density of CGR, and its predominance for existing and future space networks, as attested by its standardization under the SABR specifications or by the place it is expected to take in the future Mars communication networks as stated in the IOAG report [ioa22].

In this context, the following research question for this thesis was stated in section 1.3:

Can a routing algorithm increase scalability, network performance, and computational performance simultaneously, with respect to the state-of-the-art expectations, for both unicast and multicast bundles?

This question was decomposed into three different theses to address three specific aspects of space DTN scalability. The preliminary analysis (section 4) defined the improvement levers, and the concept chapter proposed new solutions to be resilient to the identified issues. The analysis of simulation results from section 7.5 evaluated the fulfillment of the requirements detailed in sections 2.2.1, allowing the following assessments:

/T1/ An intraregional scheduled DTN routing algorithm for unicast and multicast bundles can show enhanced networking performance and reduced computational cost regarding the SABR standard expectations for the foreseen network sizes.

Assessment: The proposed routing algorithm shows networking performance that is equivalent or better when compared with the current SABR implementations for unicast and multicast load with the test configurations. The computational pressure of the approach proved to be controlled by discarding the former hardly predictable dangerous behaviors while still being sustainable in larger scenarios if compared with the reference variant of CGR. The computational pressure was comparable to or slightly higher than CGR variants that aim to reduce the computational pressure but freed from unacceptable behavior occurrences. Evaluation of those CGR flavors on heterogeneous topologies highlighted a significant decrease in networking performance due to important contact suppressions.

/T2/ The volume management can be processed accurately with a unified and efficient algorithm compatible with the support of priority and data rate variation extensions.

Assessment: The new volume management concept proposed to accurately track time intervals of bandwidth utilization rather than inferring the transmission window from queue analysis or booked volume metrics. The approach was expected to be resilient to the current edge cases and to bring resilience while operating within future topologies where such issues are expected to be more frequent. The approach's effectiveness relies on the accuracy of the contact plan, and special dispositions were proposed to leverage heterogeneous volume management techniques if this requirement cannot be observed for each network contact. Proofs of concept were also proposed to demonstrate the flexibility of the approach for support of existing and future volume management features.

/T3/ The interregional routing for multiple actors can be supported with flexibility and reduced configuration pressure.

Assessment: To address the predictable issues associated with interregional routing support, a design redefining the interregional borders was proposed. Shifting the border from the nodes to the contacts and abstracting the interregional neighboring nodes with a simple graph theory principle allows any region node to operate with a single contact plan. The nodes can therefore be configured regardless of their regional characteristics. Given a fixed regional neighboring, single nodes can communicate with an arbitrarily high number of neighboring regions without overhead. From an operational point of view, the support for multiple border nodes simultaneously and the support for shortcuts increase the throughput and the robustness of the network. In contrast, the new border definition allows the design to decrease possible configuration tensions.

Consequently, a positive answer to the main research question can be derived, thanks to mechanisms capable of increasing the performance on three different aspects of scheduled space DTN: pathfinding, volume management, and interregional routing.

8.2 Future works

8.2.1 Next development steps

This thesis addressed known and newly discovered issues by proposing new concepts with limited coverage of administration concerns. For intraregional routing, this reduced covering can be considered negligible, as the approach is applicable to the current administration strategy deployed, for example, in ION. For interregional routing, however, the administration strategy differs slightly from the original node passageway design, as the source shall know the regional membership of the destination and attach it to the bundle. The path to this destination region shall either be known by all network nodes (i.e., the nodes shall maintain their knowledge of the interplanetary backbone structure) or attached to the bundle.

If the constraints required for operations are verified, the sources sending interregional bundles would be less impacted by single regional interface issues along the paths. Still, they must maintain interregional pathing to the destination regions they are interested in. Again,

the interregional backbone is expected to be quasi-static. Still, further investigations on that matter would be appropriate.

Pathfinding on an interregional mesh rather than following forwarding rules on the interregional tree seems like a first step towards recursive internetworking. Possible mapping with the RINA principles [Joh07] could also be further investigated.

Concerning multicast, accurate volume updates become challenging in complex topologies if node exclusion lists are associated with only a subset of the multicast destination nodes. Using a basic tree construction instead of CGR or SPSN with multipath tracking does not suffer from this problem. Further research to identify the most appropriate strategy has to be conducted.

In the context of volume management, SABR treats route expiration for single bits of information. The bundles can be arbitrarily large, and the expiration of a route is, therefore, bundle dependent if subsequent contact overlapping occurs. The bundle fragmentation strategy leverages the route expirations for maximum contact utilization, endangering end-to-end transmission due to the simplification mentioned earlier in the route expiration definition. Moreover, even if no overlapping can be observed, this strategy is viable only if the contact plan is accurate. The absence of sufficient margin or the use of average data rates instead of more accurate data rate variation handling makes those fragments more likely to be dropped due to a premature contact termination or a data rate inferior to the considered average data rate for the planned transmission timeframe.

Lastly, implementing a production-ready version could permit SPSN to be flight-tested. Python is quite adapted to research activities but rapidly shows limits regarding a complex algorithm targeting space devices, requiring all optimization and safety tools available. An implementation in C++ or Rust would therefore be welcome to integrate SPSN easily into various bundle protocol implementations for production.

8.2.2 Contact graph routing

The preliminary analysis of CGR exposes newly discovered concerning issues. Even if a possible adoption of SPSN is envisaged, the transition might be lengthy and represent a non-negligible timeframe (apart from potential SPSN democratization delays, standardization can be a long process) during which CGR will still be used for operations.

The discovered issues are of two natures. A first set of issues decreases CGR's SABR compliance. Those issues are not necessarily critically dangerous for operations but allow CGR to operate with unintended behaviors.

Yen's algorithm issues can endanger operational activities and be considered critical issues that must be fixed as soon as possible. With its hangs, Yen's algorithm behavior can be highly damageable by rendering the routing algorithm unresponsive for arbitrarily long periods and allocating memory to possible breaking points.

With such peculiarities and section 3.4 as a reference for discussion, CGR can hardly be considered an algorithm showing predictable behavior for both the expected processing time and memory pressure, as the outcome of bundle scheduling is topological and load dependent. CGR will not always compute the shortest distance path (at least for secondary metrics when using the current node-based approach) as intended by the original Dijkstra algorithm, depending on topological characteristics, and the hangs observed with Yen's algorithm renders the processing time and memory pressure highly variable.

Here follows a non-exhaustive list of suggestions that could permit relative mitigation of

the issues covered in section 4:

- Dijkstra's algorithm core should be replaced by a node graph implementation leveraging hashmaps. This would allow the detection of a route with the earliest arrival time possible. This would drastically lower the computational pressure. The hop count can still be suboptimal, but SABR compliance could be reached for at least the primary metric.
- Yen's algorithm should be dropped. The possible advantages of using Yen's algorithm are completely absent if an adaptive behavior is leveraged. The pathfinding inaccuracies could be addressed by finding the best existing routes in the set of alternative routes found by Yen. As long as Yen's main loop is paused as soon as a suitable route is found, the previous pathfinding weaknesses never get the chance to be addressed by Yen's algorithm in time. As the adaptive behavior can hardly be abandoned due to processing time concerns while being ineffective in preventing combinatory explosions and hangs, the only rational option is to replace Yen's algorithm. According to the current adaptive behavior, replacing Yen's algorithm with a single volume-aware Dijkstra search would have the same effect. This volume-aware search could still be inserted in the routing table.
- If the volume-aware Dijkstra search is deployed, the bundle size that triggered the search must be attached to the route before insertion in the routing table. During route selection of a bundle, if the size attached to a candidate route is higher than the bundle size, the route should not be selected. In the same way as SPSN, and especially for large bundles, the computation of a route (or a tree) for a given bundle size can ignore paths suitable for smaller bundles.
- The same way, all bundle-specific forwarding constraints shall be considered during route construction. This can include node exclusions, priority, expiration, or a list of destinations for multicast. Such constraints shall also be handled correctly during route selection for other bundles.

Those countermeasures are expected to bring back predictability to CGR's behavior, at least to some extent. Full compliance to the SABR's expectations seems complicated without the introduction of concepts like multipath tracking, and multicast handling is rendered more challenging by single destination route structure if compared with trees. Concerning interregional routing, implementing the contact passageway design with CGR seems like a manageable challenge, with the condition of discarding the contact overlapping constraint of SABR.

Short term, at least a subset of those modifications are required to extend the lifespan of CGR. In the long term, shifting from CGR to an alternative approach like SPSN could be valuable for effective and scalable schedule-aware bundle routing.

Bibliography

[ABB+15] Giuseppe Araniti, Nikolaos Bezirgiannidis, Edward Birrane, Igor Bisio, Scott Burleigh, Carlo Caini, Marius Feldmann, Mario Marchese, John Segui, and Kiyohisa Suzuki. Contact graph routing in dtn space networks: overview, enhancements and performance. *IEEE Communications Magazine*, 53(3):38–46, 2015.

[Ale19] Nicola Alessi. Hierarchical inter-regional routing algorithm for interplanetary networks, 2019.

[Bar74] Charles A Barth. The atmosphere of mars. *Annual Review of Earth and Planetary Sciences*, 2(1):333–367, 1974.

[BB96] Gilles Brassard and Paul Bratley. *Fundamentals of algorithmics*. Prentice-Hall, Inc., 1996.

[BBC+17] A. Berlati, S. Burleigh, C. Caini, F. Fiorini, J.J. Messina, S. Pozza, M. Rodolfi, and G. Tempesta. Implementation of (o-)cgr in the one. In *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 132–135, 2017.

[BCM+14] N Bezirgiannidis, C Caini, DD Padalino Montenero, M Ruggieri, and V Tsaoussidis. Contact graph routing enhancements for delay tolerant space communications. In *Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), 2014 7th*, pages 17–23. IEEE, 2014.

[BCMR16] S. Burleigh, C. Caini, J. J. Messina, and M. Rodolfi. Toward a unified routing framework for delay-tolerant networking. In *2016 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pages 82–86, 2016.

[BCS94] R Branden, D Clark, and S Shenker. Integrated services in the internet architecture: an overview, rfc 1633. *World Wide Web. http://www. ietf. org*, 1994.

[BCT16] Nikolaos Bezirgiannidis, Carlo Caini, and Vassilis Tsaoussidis. Analysis of contact graph routing enhancements for dtn space communications. *International Journal of Satellite Communications and Networking*, 34(5):695–709, 2016.

[BFB22] Scott Burleigh, Kevin Fall, and Edward J. Birrane. Bundle Protocol Version 7. RFC 9171, January 2022.

[BGJ+06] John Burgess, Brian Gallagher, David D Jensen, Brian Neil Levine, et al. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Infocom*, volume 6. Barcelona, Spain, 2006.

[Bir11] Edward J. Birrane. Improving graph-based overlay routing in delay tolerant networks. In *2011 IFIP Wireless Days (WD)*, pages 1–6, 2011.

[BTDT13] Nikolaos Bezirgiannidis, Fani Tsapeli, Sotiris Diamantopoulos, and Vassilis Tsaoussidis. Towards flexibility and accuracy in space dtn communications. In *Proceedings of the 8th ACM MobiCom workshop on Challenged networks*, pages 43–48, 2013.

[Bur07] Scott Burleigh. Interplanetary overlay network: An implementation of the dtn bundle protocol. 2007.

[Bur09] Scott Burleigh. Contact graph routing. *http://tools.ietf.org/html/draft-burleigh-dtnrg-cgr-00*, 2009.

[Bur18] Scott Burleigh. Bundle in bundle encapsulation (bibe). 2018.

[CBH+07] 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.

[CCS19] CCSDS. Schedule-aware bundle routing. *Consultative Committee for Space Data Systems*, 2019.

[CDCP21] Carlo Caini, Gian Marco De Cola, and Lorenzo Persampieri. Schedule-aware bundle routing: Analysis and enhancements. *International Journal of Satellite Communications and Networking*, 39(3):237–249, 2021.

[CFR18] Michael Carosino, Juan A Fraire, and James A Ritcey. Integrating scheduled dtns and tdma-based mac sublayers: Preliminary results. In *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pages 141–146. IEEE, 2018.

[CHS+22] Jacob Cleveland, Alan Hylton, Robert Short, Brendan Mallery, Robert Green, Justin Curry, Devavrat Vivek Dabke, and Olivia Freides. Introducing tropical geometric approaches to delay tolerant networking optimization. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–11, 2022.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms second edition. *The Knuth-Morris-Pratt Algorithm*, 2001.

[DBDG18] Sangita Dhara, Scott Burleigh, Raja Datta, and Sujoy Ghose. Cgr-bf: An efficient contact utilization scheme for predictable deep space delay tolerant network. *Acta Astronautica*, 151:401–411, 2018.

[DF07] Michael Demmer and Kevin Fall. Dtlsr: Delay tolerant routing for developing regions. In *Proceedings of the 2007 workshop on Networked systems for developing regions*, pages 1–6, 2007.

[DGDG19] Sangita Dhara, Chakshu Goel, Raja Datta, and Sujoy Ghose. Cgr-spi: A new enhanced contact graph routing for multi-source data communication in deep space network. In *2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 33–40. IEEE, 2019.

[DHP17] Rachel Dudukovich, Alan Hylton, and Christos Papachristou. A machine learning concept for dtn routing. In *2017 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pages 110–115, 2017.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DJ19] Olivier De Jonckère. Efficient contact graph routing algorithms for unicast and multicast bundles. In *2019 IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pages 87–94, 2019.

[DLF+17] Rémi Diana, Emmanuel Lochin, Laurent Franck, Cédric Baudoin, Emmanuel Dubois, and Patrick Gélard. Dtn routing for quasi-deterministic networks with application to leo constellations. *International Journal of Satellite Communications and Networking*, 35(2):91–108, 2017.

[DP18] Rachel Dudukovich and Christos Papachristou. Delay tolerant network routing as a machine learning classification problem. In *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 96–103, 2018.

[EB10] Daniel Ellard and Dan Brown. Dtn ip neighbor discovery (ipnd). *IETF Draft*, 2010.

[Epp98] David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

[Fal03] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, 2003.

[FDJB21] Juan A Fraire, Olivier De Jonckère, and Scott C Burleigh. Routing in the space internet: A contact graph routing tutorial. *Journal of Network and Computer Applications*, 174:102884, 2021.

[FFW18] Marius Feldmann, Juan A Fraire, and Felix Walter. Tracking lunar ring road communication. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[FLM08] Dennis Ferguson, Acee Lindem, and John Moy. OSPF for IPv6. RFC 5340, July 2008.

[FMCF18] Juan A Fraire, Pablo G Madoery, Amir Charif, and Jorge M Finochietto. On route table computation strategies in delay-tolerant satellite networks. *Ad Hoc Networks*, 80:31–40, 2018.

[FW17] Marius Feldmann and Felix Walter. Refining the ring road-delays and path lengths in a leo satellite message-ferry network. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7, 2017.

[GFPBJ16] João Gonçalves Filho, Ahmed Patel, Bruno Lopes Alcantara Batista, and Joaquim Celestino Júnior. A systematic technical survey of dtn and vdtn routing protocols. *Computer Standards & Interfaces*, 48:139–159, 2016.

[GS78] Leo J Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 8–21. IEEE, 1978.

[GY05] Jonathan L Gross and Jay Yellen. *Graph theory and its applications*. CRC press, 2005.

[HR13] Nora Hartsfield and Gerhard Ringel. *Pearls in graph theory: a comprehensive introduction*. Courier Corporation, 2013.

[ioa22] The future mars communications architecture. *Interagency Operations Advisory Group*, 2022.

[JC01] Lusheng Ji and M Scott Corson. Differential destination multicast-a manet multicast routing protocol for small groups. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 2, pages 1192–1201. IEEE, 2001.

[JFP04] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158, 2004.

[Joh07] DAY John. Patterns in network architecture: a return to fundamentals. 2007.

[Jon19] O. De Jonckère. Scalable multicast support for interplanetary networks (diplomarbeit), 2019.

[JWZS10] Zhigang Jin, Jia Wang, Sainan Zhang, and Yantai Shu. Epidemic-based controlled flooding and adaptive multicast for delay tolerant networks. In *2010 7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing*, pages 191–194. IEEE, 2010.

[KBGB08] C Krupiarz, C Belleme, D Gherardi, and E Birrane. Using smallsats and dtn for communication in developing countries. In *Proc. International Astronautical Congress (IAC-08. B4. 1.8)*, 2008.

[Kri14] Nikolai Krivulin. Tropical optimization problems. *arXiv preprint arXiv:1408.0313*, 2014.

[LDDG12] Anders Lindgren, Avri Doria, Elwyn Davies, and Samo Grasic. Probabilistic routing protocol for intermittently connected networks. *RFC Series*, 2012.

[Len20] Ricardo Lent. Evaluation of cognitive routing for the interplanetary internet. In *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pages 1–6, 2020.

[LYQ11] Mengjuan Liu, Yan Yang, and Zhiguang Qin. A survey of routing protocols and simulations in delay-tolerant networks. In *International Conference on Wireless Algorithms, Systems, and Applications*, pages 243–253. Springer, 2011.

[Mal93] Gary Malkin. Rip version 2-carrying additional information. Technical report, RFC 1388, Xylogics, Inc, 1993.

[Mar78] Etienne-Jules Marey. *La méthode graphique dans les sciences expérimentales et particulièrement en physiologie et en médecine*. G. Masson, 1878.

[MM82] Daniel G McNicholl and Ken Magel. The subjective nature of programming complexity. In *Proceedings of the 1982 Conference on Human Factors in Computing Systems*, pages 229–234, 1982.

[MM97] Gary Malkin and R Minnear. Ripng for ipv6. Technical report, RFC 2080, January, 1997.

[NBBB98] Kathleen Nichols, Steven Blake, Fred Baker, and David Black. Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. Technical report, 1998.

[OFI+07] Dirk Ooms, Nancy Feldman, Yuji Imai, Wim P. Livens, and Dr. Richard H. Boivie. Explicit Multicast (Xcast) Concepts and Options. RFC 5058, November 2007.

[RBF08] Manikantan Ramadas, Scott Burleigh, and Stephen Farrell. Licklider transmission protocol-specification. Technical report, 2008.

[RFM+21] Fernando D Raverta, Juan A Fraire, Pablo G Madoery, Ramiro A Demasi, Jorge M Finochietto, and Pedro R D'argenio. Routing in delay-tolerant networks under uncertain contact plans. *Ad Hoc Networks*, 123:102663, 2021.

[RHL06] Yakov Rekhter, Susan Hares, and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[SB07] Keith Scott and Scott C. Burleigh. Bundle Protocol Specification. RFC 5050, November 2007.

[Sha49] Claude E Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.

[SHC+22] Robert Short, Alan Hylton, Jacob Cleveland, Michael Moy, Robert Cardona, Robert Green, Justin Curry, Brendan Mallery, Gabriel Bainbridge, and Zander Memon. Sheaf theoretic models for routing in delay tolerant networks. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–19, 2022.

[SMM04] Archana Sekhar, BS Manoj, and C Siva Ram Murthy. Marvin: movement-aware routing over interplanetary networks. In *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.*, pages 245–254. IEEE, 2004.

[SNM+16] Donnie Savage, James Ng, Steven Moore, Donald Slice, Peter Paluch, and Russ White. Cisco's enhanced interior gateway routing protocol (eigrp). *Request for Comments*, 7868, 2016.

[SOBH16] Aloizio P. Silva, Katia Obraczka, Scott Burleigh, and Celso M. Hirata. Smart congestion control for delay- and disruption tolerant networks. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9, 2016.

[SPR05] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259, 2005.

[SRMS16] CC Sobin, Vaskar Raychoudhury, Gustavo Marfia, and Ankita Singla. A survey of routing and data dissemination in delay tolerant networks. *Journal of Network and Computer Applications*, 67:128–146, 2016.

[T+98] William Thomas Tutte et al. *Graph theory as I have known it*. Number 11. Oxford University Press, 1998.

[Tri13] Anshuman Tripathi. Space optimized multicast in delay tolerant networks. *International Journal of Computing and Network Technology*, 1(02), 2013.

[VB+00] Amin Vahdat, David Becker, et al. Epidemic routing for partially connected ad hoc networks, 2000.

[W+01] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[Wal20] Felix Walter. *Prediction-enhanced Routing in Disruption-tolerant Satellite Networks*. Doctoral dissertation, Technische Universität Dresden, 2020.

[Wal22] Felix Walter. A generic bundle forwarding interface. *arXiv preprint arXiv:2209.05039*, 2022.

[WBW+16] Guosheng Wang, Scott C. Burleigh, Ruhai Wang, Leilei Shi, and Yi Qian. Scoping contact graph-routing scalability: Investigating the system's usability in space-vehicle communication networks. *IEEE Vehicular Technology Magazine*, 11(4):46–52, 2016.

[WC04] Bang Ye Wu and Kun-Mao Chao. *Spanning trees and optimization problems*. CRC Press, 2004.

[WF19] Felix Walter and Marius Feldmann. Leveraging Probabilistic Contacts in Contact Graph Routing. In *IEEE Global Communications Conference (GLOBECOM)*, Waikoloa, HI, USA, 2019.

[YCCD06] Qing Ye, Liang Cheng, Mooi Choo Chuah, and Brian D Davison. Os-multicast: On-demand situation-aware multicasting in disruption tolerant networks. In *2006 IEEE 63rd Vehicular Technology Conference*, volume 1, pages 96–100. IEEE, 2006.

[Yen71] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.

[ZAZ05] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. Multicasting in delay tolerant networks: semantic models and routing algorithms. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 268–275. ACM, 2005.

[ZDE+93] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. Rsvp: A new resource reservation protocol. *IEEE network*, 7(5):8–18, 1993.