Large-Scale Space Network Simulator for Performance-Optimized DTNs

Nadia Kortas
NASA Glenn Research Center
Cleveland, OH, USA
nadia.kortas@nasa.gov

Timothy Recker
University of California, Berkeley
Berkeley, CA, USA
tjr@berkeley.edu

***Abstract*—High-rate Delay Tolerant Networking (HDTN) is a performance-optimized Delay Tolerant Networking (DTN) implementation that can provide multigigabit per second data rates in disruptive and high-latency space networks. Routing, especially in large-scale space networks, remains challenging due to network topologies that evolve over time. This paper presents a simulation tool that enables HDTN implementation testing at accelerated speeds, which is key for routing in large-scale space networks.**

***Index Terms*—High data-rate Delay Tolerant Networking, performance-optimized space networks, network simulation, DTN routing.**

I. Introduction

Communication in space environments on an interplanetary scale is challenging due to the extreme distances involved, signal propagation delays, and disrupted networks without end-to-end connection. The existing TCP/IP-based internet protocols operate on a principle of providing end-to-end communication and do not tolerate long delays and disruptions. Delay Tolerant Networking (DTN) [1] was designed to address these issues and to operate effectively in such environments achieving reliable automated network communications for space missions by using the bundle protocol which forms a store-and-forward overlay network [2].

With the significant rise in the number of satellites being sent into space, the scale of space communication networks continues to increase, and routing in these space–terrestrial systems remains challenging due to network topologies that evolve over time. At the NASA Glenn Research Center, an implementation of DTN called High-rate Delay Tolerant Networking (HDTN) has been developed with the goal of offering a solution that can scale to large, heterogeneous, interplanetary networks while maximizing performance [3]. However, the typical means for analyzing a network of HDTN nodes is slow and difficult, especially when scaling to large networks or long time periods. Emulations on virtual machines or local laboratory tests on physical machines were time consuming to configure and set up and did not scale to large numbers of nodes. To overcome these challenges, a simulation tool was developed that replicates the routing decisions that HDTN would take in an operative situation but does so in a controllable, easy-to-debug, and accelerated simulation environment. This paper presents (H)DtnSim, a simulator implemented based on DtnSim [4] by extending it to interact with the HDTN routing module. This simulator was created in OMNeT++ [5], a discrete event network simulator platform. DtnSim was built using this event driven framework to simulate scenarios efficiently at accelerated speeds, which is crucial for large-scale space networks where analysis is needed over long duration orbital periods. The structure of this paper is as follows: Section II provides a general HDTN software overview. The simulator design decisions and architecture are described in detail in Section III. Testing results from the tool for four different case studies are presented in Section IV. Section V summarizes the conclusion and simulator enhancements now in development.

II. HDTN architecture

HDTN software, which is available as open-source code [6], was designed with the goal of substantially reducing latency and improving throughput, even in constrained environments. For this reason, it adopts a parallel pipelined and message-oriented modular architecture, allowing the system to scale gracefully as its resources increase. State information is replicated between HDTN modules using ZeroMQ (ZMQ) [7], avoiding the use of shared memory methods of interprocess communication, which were found to create several bottlenecks in similar networking applications [8]. HDTN modules are defined in the following subsections; Figure 1 shows their interactions.

A. Ingress

The Ingress module intakes bundles and decodes the header fields to determine the source and destination of the bundles. If the link is available, Ingress will send the bundles in a cut-through mode straight to Egress; if the link is down or custody transfer is enabled (which involves moving the responsibility for reliable delivery of bundles among different DTN nodes in the network), it sends the bundles to the Storage module. Even if an immediate forwarding opportunity exists, Storage is always required when custody transfer is enabled. The bundle layer must be prepared to retransmit the bundle if it does not receive an acknowledgment within the time-to-acknowledge that the subsequent custodian has received and accepted the bundle.

 ISBN: 978-1-68558-035-3

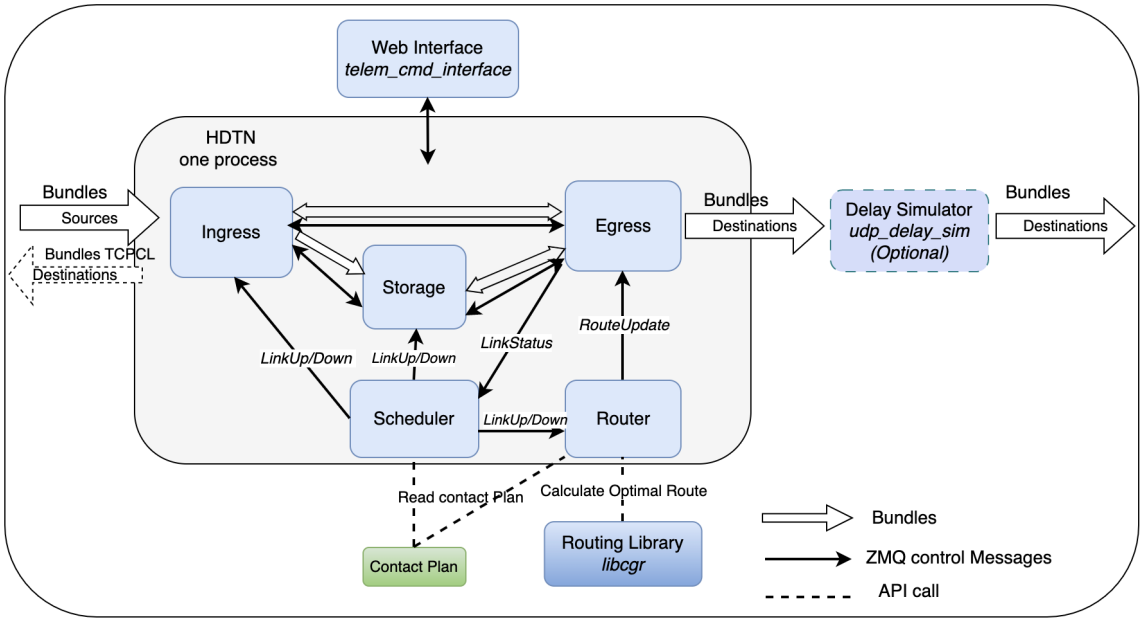

Fig. 1. HDTN software architecture and modules interactions.

B. Scheduler

The Scheduler sends `LinkUp` or `LinkDown` events with time updates to Ingress, Storage and Router. This information is used to determine if a given bundle should be forwarded immediately to Egress or stored, to recompute the route as needed if a link down would invalidate it and to keep updating the Router's own internal time before computing the optimal route. To determine the availability of a given link, the Scheduler reads a contact plan, which is a JavaScript Object Notation (JSON) file that defines all the connections between all the nodes in the network. In addition, the Scheduler dynamically handles unexpected link status changes upon receiving `HDTN_MSGTYPE_LINKSTATUS` from Egress, as well as reloading the entire contact plan upon receiving a `CPM_NEW_CONTACT_PLAN` request.

C. Storage

Storage is a multi-threaded implementation distributed across multiple disks and where custody transfer is handled. It receives messages from the Scheduler to determine when stored bundles can be released and forwarded to Egress.

D. Router

The Router module gets the next hop and best route leading to the final destination using one of the algorithms in the routing library. The router currently supports Contact Graph Routing (CGR), Dijkstra's algorithm (the default algorithm used), and Contact Multigraph Routing (CMR), which is a modified version of Dijkstra's algorithm using a multigraph structure instead of a contact graph and providing a significant performance improvement [9]. The Router then sends a `RouteUpdate` event to Egress to update its outduct to the outduct of that next hop. If the link goes down unexpectedly or the contact plan gets updated, the Router is notified, recalculates the next hop, and sends the `RouteUpdate` event to Egress so that it updates its outduct based on the new next hop.

E. Egress

The Egress module is responsible for forwarding bundles received from Storage or Ingress to the correct outduct and next hop based on the optimal route computed by Router. HDTN uses an event-driven approach based on ZeroMQ pub-sub sockets for sending unexpected link updates and contact plan changes. When the connection changes unexpectedly, Egress will send a `LinkStatus` update message to the Scheduler, which triggers it to send a `LinkDown` or `LinkUp` event to Ingress, Storage and Router to determine whether bundles should be stored or the route needs to be recomputed.

F. Web Interface

The Web Interface displays data rates graph and bundles statistics for network troubleshooting. It's also used for updating configuration, routes, and contact plans.

III. HDTN Simulator Design

Analyzing the behavior of a complete network of HDTN nodes poses some challenges. Unit tests can be used to assess individual modules in HDTN and integrated tests can assess how modules combine to implement HDTN node behavior. However, assessing the behavior of an HDTN network has traditionally required running tests on physical machines or emulations on virtual machines. Laboratory tests of up to

 ISBN: 978-1-68558-035-3

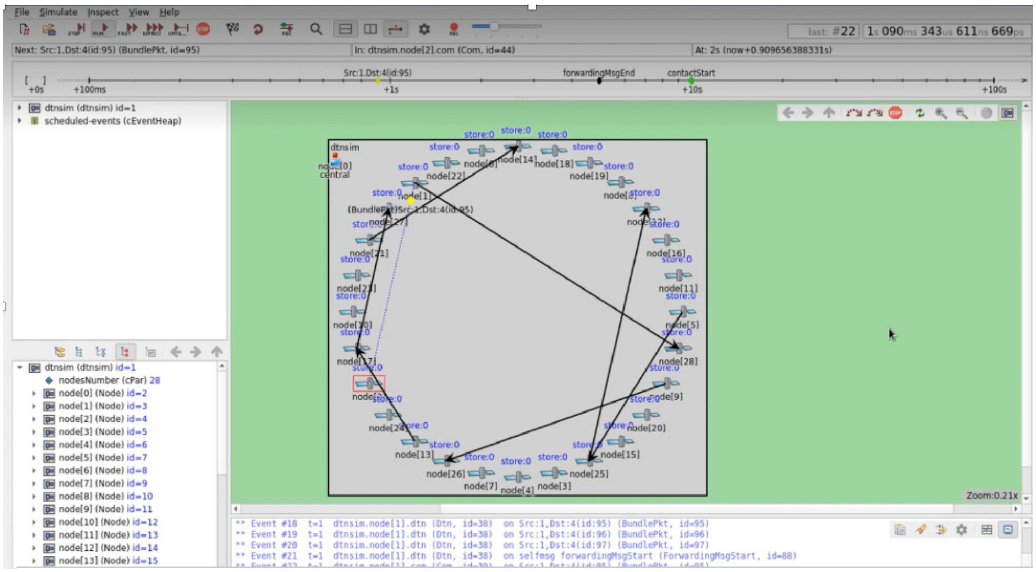

Fig. 2. (H)DtnSim Graphical User Interface (GUI)

ten nodes have been run, but a major design goal of HDTN is to achieve performance and stability that can scale to a large, interplanetary network. Given this goal, existing network analysis methods have two main problems:

- Setting up and configuring tests can be slow. This problem becomes more significant as tests scale to more nodes, contacts, and data transmitted.
- Running tests can be slow. This problem also grows as tests scale to longer time periods.

This section explores simulation as a potential solution to these two testing difficulties. A DTN simulator should exhibit six desirable properties:

(A) **A**ccuracy
(R) **R**un-time acceleration
(U) **U**tility
(D) **D**evelopment-time acceleration
(M) **M**aintainability
(S) **S**calability.

The first three properties are essential for the purpose of HDTN simulation; without them, there would be no reason to simulate HDTN instead of running nodes in the laboratory. Property A means that given the same transmission plan, contact plan, and time-varying network topology, a simulated DTN should produce the same results as a real DTN. R means that simulations should run in an accelerated simulation time instead of real time, i.e., a simulation spanning one hour of the contact plan should take less than one hour of real time to run. U means that the simulator must produce some output that can be processed—either by a machine or a human—and the output should yield some insight about the behavior of the DTN. D and M refer to the man-hour cost of developing simulation scenarios and of maintaining the simulator. S means that as one varies the size of parameters—such as number of nodes, number of contacts, time elapsed, number of bundles sent, and size of bundles sent—all previous properties continue to hold.

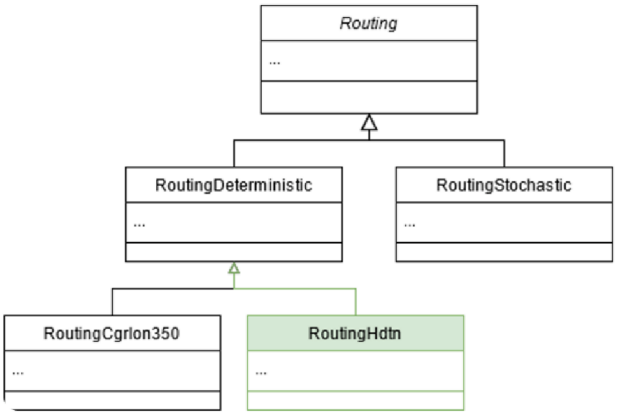

Fig. 3. DtnSim Routing class hierarchy with methods omitted. All Routing models implement the abstract Routing class and RoutingHdtn imitates the RoutingCgrIon350 class.

The approach taken to simulating HDTN in this paper is an extended version of DtnSim, referred to here as (H)DtnSim. DtnSim is a simulator for DTNs with a special emphasis on analyzing routing. It exhibits many of the desirable properties described previously in this section, including accelerated execution in non-real time (R), ease of configuration with short and simple .ini files that can be edited in text mode or using a graphical interface (D), automatic generation of metrics and diagrams for network flows and network topologies (U), and user-friendly interfaces for the visualization and control of simulation scenarios and the analysis and plotting of metrics (U). The (H)DtnSim user-friendly GUI shown in Figure 2 is a key element to gaining insight into complex time-evolving topologies.

 ISBN: 978-1-68558-035-3

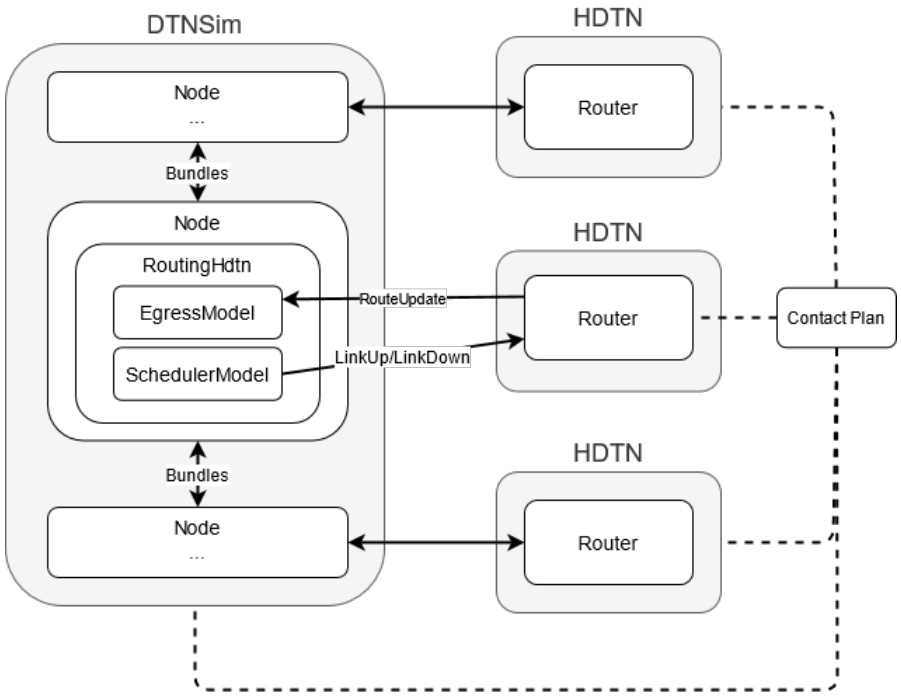

Fig. 4. Illustration of the architecture of (H)DtnSim showing three real HDTN nodes communicating via ZMQ messages with simulated nodes in DtnSim. A shared contact plan must be supplied by the user but HDTN configuration is largely generated by the simulator.

This approach is viable because DtnSim is written in a modular, object-oriented fashion that allows users to select different routing models for their simulations and enables developers to easily implement new routing models as subclasses. It relies on a class hierarchy starting from the `Routing` class and uses virtual methods throughout the code to enable extension through inheritance. Extending the base `Routing` model with subclasses enables a developer to create DTN nodes with custom behavior that can take full advantage of all the features and functionality of DtnSim. Thus, HDTN simulation is achieved by implementing a `RoutingHdtn` class as shown in Figure 3.

Two notable subclasses include `RoutingCgr Model350` and `RoutingCgrIon350`, each of which exhibits distinct and instructive approaches to simulation. The former implements a simplified version of Interplanetary Overlay Network (ION)'s routing logic using the abstractions of DtnSim. The latter can be understood as an interface gluing together DtnSim code and actual ION flight code; ION support in DtnSim is implemented by taking a subset of ION (namely the part that decides when and where bundles are forwarded or stored), compiling it into DtnSim, and calling it from the `RoutingCgrIon350` class. These two classes represent alternative ways to answer the question: "What would ION do?" After determining the answer, they replicate this action in the simulator. The `RoutingCgrIon350` class answers the question by actually running ION in accelerated simulation time and inspecting the result.

Although this approach is clever, its implementation has one notable limitation: copying ION code and compiling it into the simulator requires the simulator to be updated every time ION changes to stay current with ION. This has left ION support in DtnSim frozen at ION version 3.5.0, even though the latest version is 4.1.1. This Maintainability limitation is too costly for an HDTN simulator to embrace, given that HDTN—especially the functionality of the Router—is under active research and development. Over the ten weeks during which the research for this paper was conducted, HDTN was enhanced with several additions, including a CGR library written in C++, an implementation of the CMR algorithm, and features for time-tracking and route re-computation. Additionally, during this time, developers were researching a routing approach based on Spiking Neural Networks (SNN) using estimations and observations of the network congestion and loss, neighbor discovery and other routing enhancements.

Considering these maintainability concerns, (H)DtnSim takes the same general approach as `RoutingCgrIon350` while addressing the Maintainability issue by applying classical principles of engineering and object-oriented software: information hiding and restricting interaction between entities to limited public interfaces. HDTN itself is written in a modular, object-oriented fashion since it consists of five modules interacting through asynchronous message passing. Thus, instead of embedding HDTN within DtnSim, the authors chose to extend DtnSim with the ability to talk to HDTN: an HDTN Router process runs for each DtnSim node, communicating using the messaging protocol of HDTN over ZMQ sockets, as shown in Figure 4. Here are the steps for per-node Router initialization:

1. Map the EID of the node to a pair of unique port numbers.

 ISBN: 978-1-68558-035-3

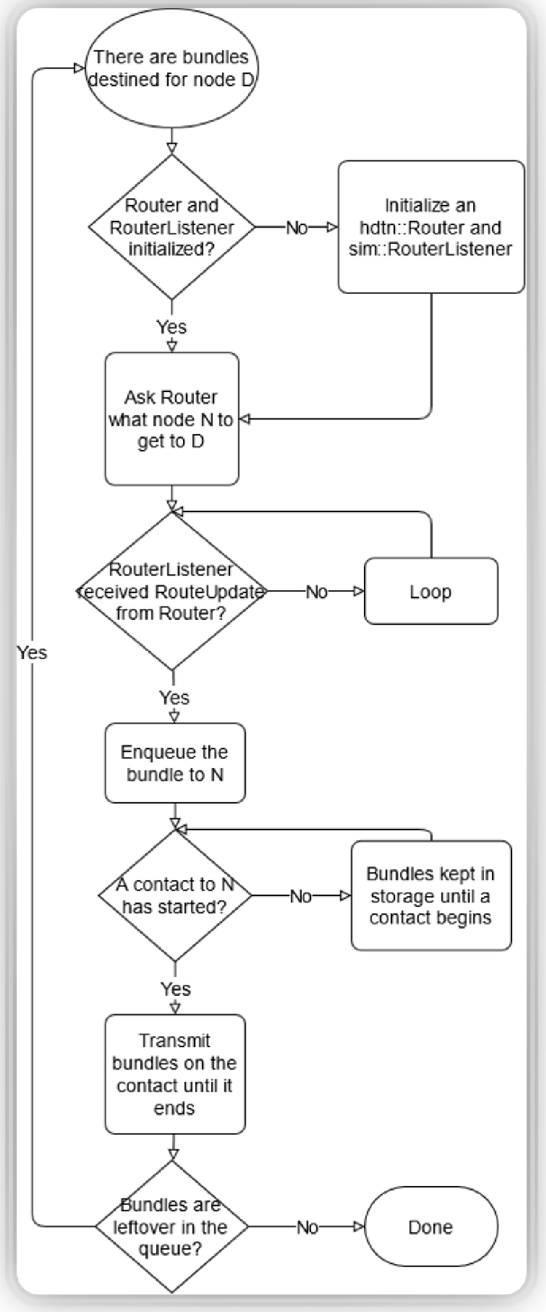

Fig. 5. Flow diagram of steps HdtnRouting takes to decide where bundles go.

2. Generate an HDTN configuration file that binds the Scheduler and Router to the generated port numbers on the localhost.
3. Run an hdtn-router executable installed on the user's system using the configuration file generated in the previous step.
4. Connect objects of types SchedulerModel and EgressModel to the Router.

The flow diagram in Figure 5 shows the steps HdtnRouting takes to decide where bundles go. The SchedulerModel sends `LinkUp`/`LinkDown` messages to the Router to keep the Router's internal notion of time synchronized with the simulator and notify it of relevant network topology changes. The EgressModel listens for `RouteUpdate` messages published by the Router and uses the Router's decision to replicate the equivalent action in DtnSim. Combined with the event loop of DtnSim facilitated by OMNeT++ messages, the SchedulerModel models the required functionality of an HDTN Scheduler module; combined with the bundle forwarding threads and the event loop, the EgressModel models the functionality of an HDTN Egress module. In other words, these two models are simplified versions of HDTN modules that stand in for the actual modules and mimic their behavior.

IV. Testing and Results

In this section, a series of simulation scenarios illustrates how this implementation exhibits the six properties desired, making it a simulator suitable for analyzing HDTN.

A. Simulator vs Runscript

To test the accuracy of the simulator, two routing test cases from the HDTN source code have been mimicked in a simulation scenario. The first is copied from a shell run script and involves one node sending bundles to two other nodes. The second is the "routing test" contained in the Linux scripts directory of the HDTN source code and involves four nodes in a network where Node 1 sends bundles to Node 4. Both tests use the same contact plans used by the HDTN test cases and produce approximately the same outcomes as summarized in Table I. For both the runscript and simulator the actual number of bundles delivered is approximately as expected and matching but there is a 1% discrepancy. In the simulator precisely 3800 bundles are delivered while in the runscript anywhere from 3834-3840 bundles are delivered with some indeterminacy. The cause of this is under investigation but there are a couple possible explanations. First, it may be that there is some imprecision in the bundle generator used in the runscript such that it does not produce exactly 100 bundles per second. Second, it may be that because the bundle generator and HDTN's Scheduler don't share a clock there is some asynchronization such that the bundle generator produces 37-40 bundles before the Scheduler is fully running and tracking time. In contrast, in DtnSim both the application layer and the underlying DTN layer share a single notion of time and are tightly synchronized. DtnSim supports the use of random variation in parameters that could allow more accurate description of the real HDTN scenario within the simulator with some additional effort. However, both of these potential effects might be heavily system dependant and difficult to quantify. Despite these nuances, this test demonstrates that A holds within a small margin of error.

Additionally, the results for the run script test detailed in Table I show that the simulation versions of the tests have a lower development time and run time. The run time can easily be measured with a physical or software stopwatch. Development time is difficult to measure precisely but as a proxy one can look at the total number of Source Lines Of Code (SLOC) that must be written to implement a scenario.

 ISBN: 978-1-68558-035-3

TABLE I
COMPARISON OF SIMULATION AND RUNSCRIPT PERFORMANCE

	Simulation	Runscript
Routes Found	$10 \rightarrow 2$, $10 \rightarrow 1$	$10 \rightarrow 2$, $10 \rightarrow 1$
Actual Bundles delivered	3800	3840 ± 6
Config lines (SLOC)	13	158
Run time (s)	1	73 ± 3
Discrete events	34558	NA

B. CGR vs CMR

As stated in Section III, a major design goal of the simulator is to abstract internal implementation details of HDTN so that HDTN can continue to develop without requiring updates to the simulator. This goal was put to the test by running two different branches of the HDTN source code in the simulator. The first branch used a CGR version of Dijkstra's algorithm to compute the best routes; the second used a CMR algorithm. These algorithms have different definitions, run-time complexities, and implementations, yet the simulator worked equally well with either branch. HDTN implementations can be swapped easily within the simulator, requiring no changes to simulator source code and only a one-line configuration file change. Thus, M holds. (H)DtnSim can deliver this flexibility and Maintainability under two conditions: 1. The architecture of HDTN cannot change, including changes to the structure or semantics of the messages passed between HDTN modules 2. The user interface to HDTN must remain backwards compatible, i.e., the command line interface must continue to support the syntax and options that the simulator uses to run HDTN.

If either of these conditions is unmet, the simulator might require updates to continue working with the latest version of HDTN. Architectural and interface updates amount to changing the design of HDTN. Given that the simulator models the behavior of HDTN, it is hardly surprising that such design changes necessitate updates to the simulator that models it. However, as long as the design of HDTN remains stable, changes in implementation details will not negate the accuracy of the simulator's model. This approach comes with a few noteworthy requirements or limitations:
1. (H)DtnSim must implement some of the messages that HDTN uses.
2. (H)DtnSim must use only these messages to get information about the running HDTN Router; dissecting the internal state of the Router compromises the abstraction layer.
3. (H)DtnSim must interpret the semantics of these messages in a way that is equal to the HDTN interpretation or at least similar enough to replicate HDTN's behavior.

Because of the flexible nature of (H)DtnSim, performance enhancements can be made to HDTN without requiring a change to the simulator. Thus, in addition to facilitating Maintainability, this property of the simulator supports HDTN's core performance mission.

C. HDTN vs ION

The simulator has been used to identify useful enhancements to HDTN by running simulation scenarios for both HDTN and ION and comparing the results. As the flow diagrams in Figures 6 and 7 indicate, ION can transmit 1,728,000 bytes over the last contact in the simulation for a total of 7,680,000 bytes transmitted; in HDTN, those bytes remain stranded on node one, resulting in a bundle loss rate of 22.5 percent. After inspecting the HDTN source code, it was determined that this difference in packet delivery rate was primarily the result of the way the HDTN Router handled time and changes in the network topology. In particular, the Router formerly did not update its time from the initial time of the contact plan, nor did it recompute routes when a link down event should make a route it previously computed invalid.

The effect of this problem is difficult to quantify generally because it heavily depends on the precise contact graph and network traffic. Thus, the difference in bundle delivery rates between ION and HDTN resulting from this issue could be 0%, 100%, or anything in between. However, bundle delivery rate is an important performance metric and the situation that produced this discrepancy between ION and HDTN is quite plausible in realistic network topologies and workloads; all it requires is that the Router computes a route for some bundles and one of the links in this route later goes down while previously routed bundles are still awaiting delivery. The discovery of this problem and the clarity of its illustration is a strong testament to the Utility of (H)DtnSim.

As a result of this discovery, enhancements to HDTN time-tracking and route computation were made, resulting in the changes illustrated in the network flow diagrams of Figures 8 and 9. These enhancements were implemented with architectural changes to HDTN, which required updates to the simulator that are under development. This is a good example of the limits of property M: the simulator is resilient to internal implementation changes but can require significant updates when changes are made to HDTN's ZMQ sockets, message structure, message semantics or CLI options.

D. Scaling to Large Networks

For this section, the simulations were constructed with four fictitious ground stations, at Albany, NASA Glenn Research Center, University of California Berkeley, and Guam based on data from Starlink satellite orbits. With each simulation lasting for 24 hours (86400 seconds), and counting the ground stations as nodes, the contact plans consisted of 14, 54, 104, and 204 nodes corresponding to 368, 7186, 28162, and 109330 contacts, respectively.

TABLE II
SIMULATION RESULTS USING THE FOUR LARGE CONTACT PLANS

Nodes	14	54	104	203
Contacts	368	7186	28162	109329
Time (s)	5	7	15	94
Discrete events	611460	2157761	3098460	6658661

 ISBN: 978-1-68558-035-3

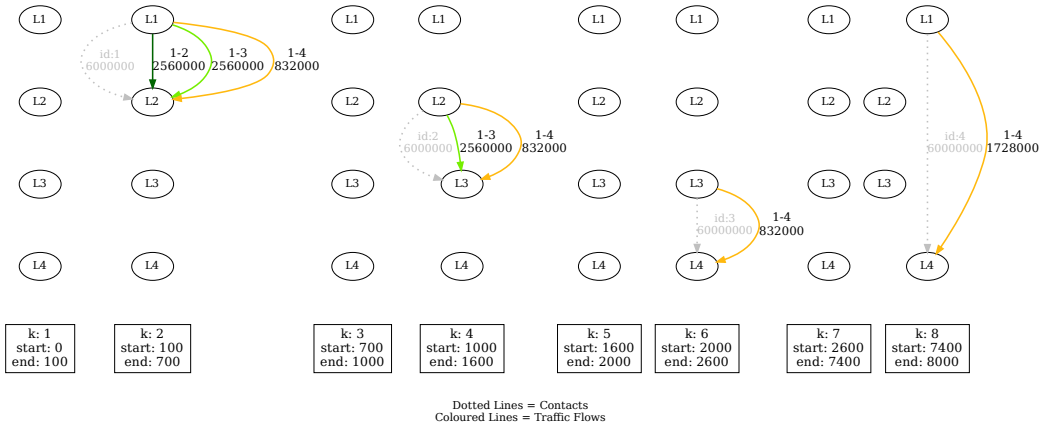

Fig. 6. A data flow diagram for a network of four nodes running ION. All links are utilized in this scenario for 100% bundle delivery rate.

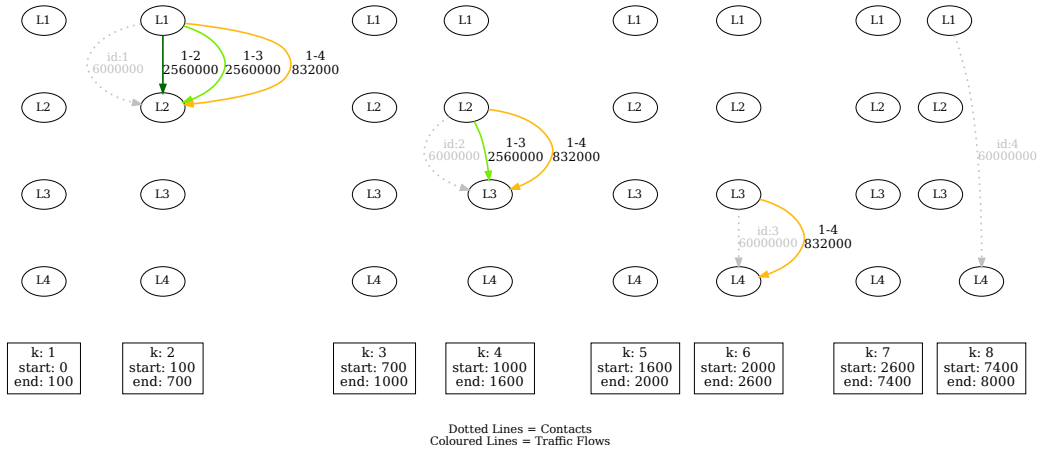

Fig. 7. A data flow diagram showing how HDTN handled the same scenario shown in Figure 6. The link from L1 to L4 in the last network topology is not utilized, resulting in bundle loss.

The data in Table II and Figures 10 and 11 show the results of a simple, fixed scenario run with the four contact plans described. The network is populated by nodes connected through intermittently connected, gigabit-rate links. For 20 seconds of simulation time, one ground station sends 1907 bundles per second consisting of 65535 bytes to another ground station, using routes with two hops via satellite. These values are selected to saturate the gigabit links with bundles equal in size to the maximum size of a TCP packet. In each of the four runs, this situation is constant and only the number of nodes and contacts varies.

These results—combined with preliminary inspection of the effects of bundle count and route recomputation—suggest that the duration of a simulation depends on (a) the number of discrete events in the simulator and (b) the time spent running the HDTN Router. The former is mostly—aside from a small extra startup overhead—independent of the number of nodes, length of simulated time, and size of bundles sent; it depends instead on the number of contacts and number of bundles sent. However, in a dense network topology (graph) the number of contacts (edges) will increase quite rapidly with the number of nodes (vertices). Thus, the results of x, y, z show a steep jump in simulation runtime in the step from 104 to 203 nodes since the topologies in this scenario are somewhat dense. This issue should look familiar to readers aware of the challenges resulting from the amount of scheduling information required to use CGR in large networks. However, a sparse network topology would not face this issue.

On the other hand, part of the time spent running the HDTN Router depends on the implementation and computational complexity of HDTN's routing algorithm. This part indicates nothing about the performance of the simulator itself. However, time spent running the HDTN Router also depends on the simulator implementation. Some performance optimizations

 ISBN: 978-1-68558-035-3

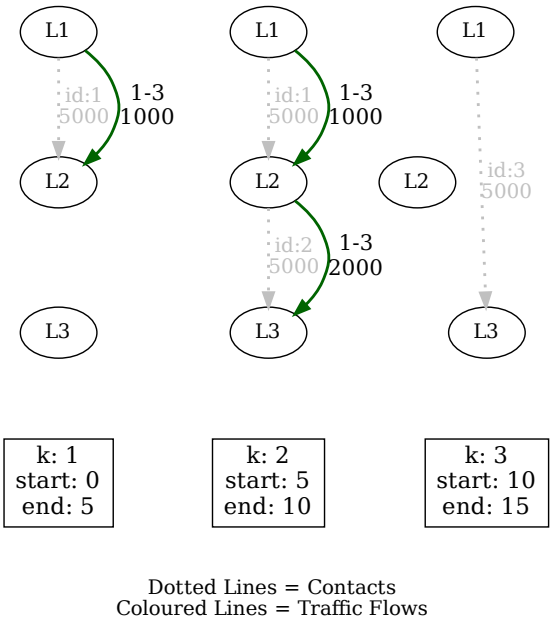

Fig. 8. Data flow diagram before Router changes. A link from L1 to L3 is available in the third network topology but is not utilized to transmit bundles because the Router did not track time and thinks the next hop should still be L2.

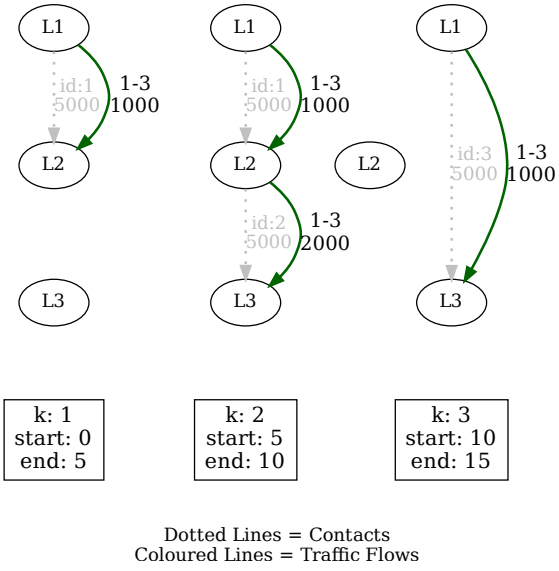

Fig. 9. Data flow diagram after Router was updated to track time. Unlike in Figure 8, the link from L1 to L3 is properly utilized to deliver bundles.

have been implemented to address this, such as caching results from the HDTN Router. Others, like more efficient

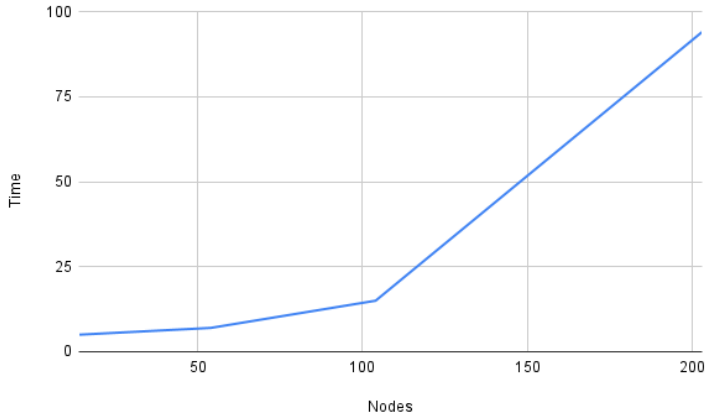

Fig. 10. The most significant factor in simulator performance when scaling to large networks is the time (in seconds) required to run a simulation versus node count.

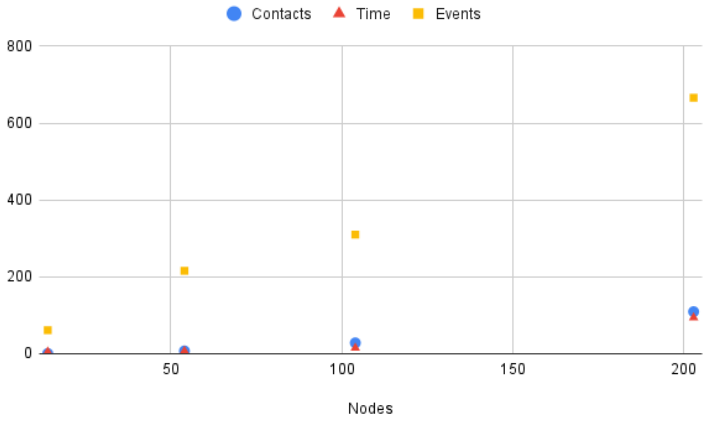

Fig. 11. Contacts expressed in thousands, time in seconds, and events in tens of thousands. Scaled to fit on one plot to show the corresponding slopes.

use of Linux threads and processes, are under development. The simulations in this section are run in a VirtualBox VM on a commodity laptop with no setup tuning. As such, the numbers indicate little about expected runtimes for simulations running with plentiful resources. They do, however, give some idea of how performance persists or degrades with larger networks. The results suggest that (H)DtnSim is successful in *S*caling to networks with many nodes. While simulation performance could be improved, it may be limited by the inherent complexity of the algorithms and the rate of growth for contacts when nodes are added.

V. Conclusion and Future Work

From this research, one can see that a simulator—and the (H)DtnSim simulator in particular—is a satisfactory solution to some problems with testing in HDTN. In addition to the expected and intended outcomes, the simulator facilitates exploration and discovery of HDTN behavior through rapid development and deployment of tests. This process

 ISBN: 978-1-68558-035-3

produced enhancements to the HDTN Router's handling of time, improving bundles delivery rate. The design of HDTN has proven highly modular, flexible, and extensible, making enhancements like these easy to incorporate and interaction with the simulator seamless and maintainable.

(H)DtnSim has been made available publicly to the DTN community under the branch "support-hdtn" [11], which can be found in the official DtnSim repository [4].

Future versions of (H)DtnSim will support the multi-destination routing enhancements. It will also support opportunistic links and unexpected link disruptions. Doing more extensive testing by running simulations on more powerful servers will strengthen the evidence for the claims made about the simulator and provide more insights about HDTN performance.

ACKNOWLEDGMENT

The authors would like to thank the NASA Space Communications and Navigation (SCaN) program. Support from the rest of the HDTN team and guidance from the DtnSim developers, including Juan Fraire and Pablo Madoery, is much appreciated.

REFERENCES

[1] V. Cerf et al., "Delay-Tolerant Networking Architecture," RFC 4838, IETF: Fremont, CA, USA, 2007. [Online]. Available: https://datatracker.ietf.org/doc/rfc4838/. Accessed March 24, 2023.

[2] K. Scott and S Burleigh, "Bundle Protocol Specification," RFC 5050, IETF Network Working Group (2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5050. Accessed March 24, 2023.

[3] A. Hylton et al., "New Horizons for a Practical and Performance-Optimized Solar System Internet," IEEE Aerospace Conference, 2022. [Online]. Available: https://doi.org/10.1109/AERO53065.2022.9843598. Accessed March 24. 2023.

[4] J. Fraire and P. Madoery. (2019). DtnSim Official Repository. [Online]. Available: https://bitbucket.org/lcd-unc-ar/dtnsim/src/master/. Accessed March 24, 2023.

[5] A. Varga. "The OMNeT++ Discrete event simulation system," in Proc. European Simulation Multiconference, Prague, Czech Republic, 2001, pp.1-7.

[6] B. Tomko, N. Kortas, R. Dudukovich, and B. LaFuente. (2021). HDTN Official Repository. [Online]. Available: https://github.com/nasa/HDTN. Accessed March 24, 2023.

[7] P. Hintjens. (2020). ZeroMQ: Messaging for Many Applications [Online]. Available: https://zeromq.org/. Accessed March 24, 2023.

[8] A. Hylton, D. Raible, and G. Clark, "On the Development and Application of High Data Rate Architecture (HiDRA) in Future Space Networks," AIAA 2017-5415. [Online]. Available: https://arc.aiaa.org/doi/pdf/10.2514/6.2017-5415. Accessed March 24, 2023.

[9] R. Kassouf, "Contact Multigraph Routing: Overview and Implementation," presented at the 2023 IEEE Aerospace Conference, Big Sky, MT, March 4-11, 2023, Paper 4.0906.

[10] T. Recker. (2022). HDTN Example Simulation [Online]. Available: https://bitbucket.org/lcd-unc-ar/dtnsim/src/master/. Accessed March 24, 2023.

[11] T. Recker. (2022). (H)DtnSim Branch. [Online]. Available: https://bitbucket.org/lcd-unc-ar/dtnsim/src/support-hdtn/. Accessed March 24, 2023.

 ISBN: 978-1-68558-035-3