Bundle Protocol version 7 Implementation with Configurable Faulty Network and Evaluation

Aidan Casey, Ethan Dickey, Jihun Hwang, Sachit Kothari, Raushan Pandey, Wenbo Xie[§]
Department of Computer Science, Purdue University, West Lafayette, Indiana, USA
Email: {casey39, dickeye, hwang102, kothar13, pandey72, xie401}@purdue.edu

***Abstract*—The Bundle Protocol (BP) is a key component that enables delay/disruption-tolerant networking (DTN), an overlay network architecture that facilitates communication in challenging environments with intermittent connectivity. Previous works on DTNs were largely on analyzing or optimizing DTNs instead of studying BP alone itself; hence, whether or not BP is still at an experimental stage must be studied. This work begins by presenting a lightweight implementation of BP Version 7 (BPv7) created by implementing only the required portions of RFC9171, with a new convergence layer that simulates expected and unexpected disruptions for testing purposes. Our implementation is lightweight enough to be easily extendable for additional tests and simple enough to be used for educational purposes. Some preliminary, lightweight experiments indicate that BPv7, even with only the required parts in RFC9171, can serve its purpose and still ensures essential functionalities. It tolerates disruptions and infinitely long delays well, as intended. Moreover, it handles large data dumps and floods of packets well, as long as they are infrequent. In the course of our implementation and experiments, we identified potential architectural, specification, and deployment-related flaws of BP, and suggested solutions or directions toward them from the perspective of software engineering and network algorithms.**

***Index Terms*—Bundle Protocol, Delay Tolerant Network, Store-and-forward Networking, Convergence Layer, Bundle Layer**

I. Introduction

Thanks to the development of wireless and mobile networks integrated into our daily lives, communication does not seem to be a technical issue anymore. Unfortunately, just as the 5G network only covers the major cities in the United States currently, not every region on Earth gets to benefit simultaneously from such technological advancements. This is not entirely the fault of the distribution of technology, however. Computer networks under extreme environments are inevitably subject to performance barriers such as intermittent connectivity, long latency, and asymmetric data rate. Under traditional networks, such as the celebrated TCP/IP-based Internet widely used today, those effects could result in imperfect end-to-end connections between two end hosts, packet losses, and increased metadata in the network.

Delay/disruption tolerant network (DTN) is an overlay network that is designed to provide stability and resiliency against extreme conditions by transporting data packets *hop-by-hop* in *store-and-forward* manner. The concept of DTN was formalized by Kevin Fall and first appeared in the general public from his SIGCOMM '03 paper [1] as a generalized version of the Interplanetary Internet (IPN) architecture that existed back then, and it is now a building block of IPN and serves as the foundation for many other 'challenged Internets' lacking continuous connectivity [2]. RFC 4838 [3] formally defines DTN, and describes its basic architecture and requirements.

In layman's terms, DTN architecture is similar to the Internet Protocol suite (TCP/IP), except there is a new layer called the *bundle layer*, usually added between the application and transport layers. Readers unfamiliar with DTN and its network architecture can refer to relevant RFCs [3]–[5] and recent surveys/textbooks such as [6]. Bundle layers can connect 'disconnected' end hosts by generating then storing and forwarding *bundles*, the primary data unit (PDU) used in the bundle layer analogous to datagrams in the network layer and segments in the transport layer.

The communication protocol designed for the bundle layer is called *bundle protocol* (BP), and the most recent version of bundle protocol is *Bundle Protocol version 7* (BPv7) which is specified in RFC 9171 [5]. To summarize it briefly, BP largely consists of three sublayers: *application agent* (AA), *bundle protocol agent* (BPA), and *convergence layer adapters* (CLAs). AA functions as a 'secretary' of the BPA and socket between application and bundle layer; it is responsible for processing and handling *administrative records* for the BPA, as well as receiving application data from the BPA and forwarding it to application layer. BPA is the 'core' of BP, generating bundles based on data received from the application layer through the AA and sending/receiving bundles to/from the CLA (or convergence layer, if no CLA). CLA sends and receives bundles from the transport layer, and informs the BPA of necessary information regarding network status. See RFC 5050 [4] (BPv6) and 9171 [5] for more details.

A. Problem

DTN is a fairly well-established concept and, although limited, is currently being used to transfer user data between International Space Station (ISS) and ground stations [7]–[9]. However, outside of that, there have not been many practical applications of DTN, even for small-scaled terrestrial ones, due to the lack of understanding of DTN and BP on many fronts. Several studies were conducted to identify barriers that prevent the complete integration of DTN into our real-life applications. As pointed out and studied in multiple recent works, one such

[§]Authors are ordered alphabetically, following the Hardy-Littlewood axioms of collaboration used in theoretical computer science society.

barrier is the lack of our theoretical understanding of DTN itself, such as mathematical modeling [10], scalability [11], [12], and routing/scheduling [13]–[15].

In contrast, the problems this paper aims to address are more on the practical, architectural, and implementation aspects of BPv7 as defined in RFC 9171 exactly. BPv7 is supposedly an improved version of BPv6 that is not (directly) backward compatible due to many factors such as encoding methods and error-detecting codes [16]–[18]. However, to our current knowledge, there have not been many works discussing the difference between the two versions in terms of performance, let alone the feasibility and practicality of BPv7 itself, or what aspects of BPv7 in RFC 9171 need to be modified or improved [13], [19]. We should also point out that many BPv7 implementations that are currently available to the public such as [20]–[23] were either written in comparatively unpopular languages, or more importantly, were written when RFC 9171 was a draft which is now expired and hence not completely free from errors or discrepancies. For example, one implementation appears to assume that BPA and AA communicate by sending bundles to each other, but per RFC 9171, they must be communicating using application data units (ADUs). Other implementations (such as [24]–[27]) that are more geared towards simulating a (large-scaled) DTN or evaluating/optimizing the performance of the applications and network overall, treat BP more as a 'black box' than a portion of a network that should be extensively studied. It is not very clear whether the delay/disruption tolerance feature of DTN is attributed to the protocol primarily, or from some other aspects of the network.

B. Contribution

We begin by presenting a nearly complete implementation of BPv7[1] as defined in RFC 9171 using Java, one of the most popular programming languages today. Our implementation is rather 'lightweight'—easy to reverse-engineer and extend as needed—and is specifically designed to study and analyze BPv7. For these reasons, it will contain only the parts that are imperative to get a small-scale DTN functioning and only the parts that are strictly required per RFC 9171; for example, we did not take security into consideration. Note again that this paper only aims to study the architecture of BPv7, not to construct and evaluate a candidate DTN for practical usages. The only required portion of BPv7 that we did not implement was the CBOR data format. In favor of comprehension and debugging the network, we chose to use JSON instead.

We conducted experiments[2] on our BPv7 implementation by simulating it on a tiny-scaled software-defined network (SDN) built using Mininet [28] and Open Network Operating System (ONOS) [29]. We designed a de facto convergence layer called *Disruption-TCP* (DTCP) that allows BPv7 to communicate effectively with TCP in our (Mininet-simulated) transport layer and mimic disruptions that can be present in DTN by generating random delays and drops. Section II-A will provide more details regarding DTCP. We perform several simulations of our implementation based on configurable scenarios. See Section III for more information about the particular parameters used in testing and our related findings.

We also evaluate the current specification of BP and DTN (specified in RFCs) from the perspectives of software engineering and network architecture by presenting a set of existing architectural deficiencies that need to be addressed and missing features and specifications that can be critical to the future deployment of DTN with BP (Section IV). We then conclude with suggestions for future works (Section V-A).

II. Implementation

A. DTCP

Disruption-TCP, or DTCP, is the BPv7 convergence layer created for this research that simulates random disruptions, making our network a 'configurably faulty' network. It works by taking a series of configurable length *timeframes*, measured in milliseconds, and generates two unique seeds based on the current timeframe and the connection between any two hosts. It then uses a pseudo-random number generator with this seed to determine whether there is an expected disruption and/or unexpected disruption. This allows the connection to go down on both sides simultaneously (expectedly or unexpectedly).

1) Expected Disruptions: An *expected* disruption is any disruption that both nodes are aware of ahead of time, and know not to send any bundles across the connection during that period. An example of an *expected* disruption in the classical sense would be two nodes communicating in space, with a large celestial body occasionally crossing the path of the connection and disrupting that traffic. This disruption would be calculable ahead of time, so is considered expected programmatically. In our particular implementation, we chose to randomly simulate these expected down times in a way that both sides of the connection were aware that these were expected down times and would be down simultaneously, but that the down times themselves were not explicitly regular. A different design choice could be easily implemented. It is important to note that we assume the convergence layer stores the information about regular disruptions but that it is the responsibility of the BPA to handle those regular disruptions.

2) Unexpected Disruptions: An *unexpected* disruption occurs when some other disruption, such as a large amount of interference, a smaller body intersecting an EM wave connection between space hosts, or any other unpredictable event occurs over a period. The sending host has no idea that this happens, so the receiving host instead determines if this bundle transmission should be successful by the same pseudo-random process (different parameters) as for expected disruptions. When DTCP receives a bundle, but before passing it to the BPA, the DTCP layer decides if this bundle unexpectedly drops and if so drops it after completing the receive.

3) Alternative Designs: Potential alternatives could allow for more control of how disruptions occur, such as separate time frame windows for the two kinds of disruptions, or even

[1]Our implementation and setups for the testing environment are available in this GitHub Repository: https://github.com/etdickey/BPv7Java.

[2]Demo available in this link (YouTube): https://youtu.be/aika4nRm7wM.

the ability to have multiple configurable sub-types of each with different lengths and frequencies. However, this all costs additional overhead to the process and could potentially affect other useful measurements of performance, and as such it was determined that the two types of disruptions implemented would be sufficient for this work, where a particular length and frequency of each of the two classes of disruptions can be studied independently with less impactful overhead. Regardless, as discussed in Section IV, the requirements of a convergence layer for BPv7 are vague, and as such there is room for many potential alternatives. This solution merely has some desirable qualities and was built with simplicity in mind.

B. *Mininet*

The implementation involved the use of three virtual hosts, each performing a unique role. Host A (or Node A) is the sending node, which sends traffic to Host B which is the receiving node (i.e. $A \rightarrow B$), although traffic was also sent in the opposite direction ($B \rightarrow A$). In between Host A and Host B, there is the forwarding node, Host F. All traffic from Host A first goes through Host F, and all responses from Host B, which were mostly Status Reports about bundles sent by Host A also go back through Host F. The forwarding node Host F also sends a status report when it forwards a bundle sent through it (due to the hop-by-hop reliable delivery requirement of DTN – not BPv7). Many more complicated networks can be made through the Mininet network configuration and other configuration files, without modifying much or any of our implementation. An example transmission can be found detailed in our GitHub repository.

C. *Configuration*

For each simulation, there are a number of parameters our implementation includes. Some are built for further extension, such as certain internal queue sizes, the number of threads for receiving bundles, the maximum number of connections the server accepts at once, etc. These were kept constant within our testing.

1) Routing and Name Lookup: Routing is something that is not specified in RFC 9171, and IDs are simply URIs [30], so also need to be converted into the underlying address scheme. For this reason, and due to its status outside of the scope of this project, routing was done via configuration with hard coded routes for a given destination ID. For more information about the potential flaws in BPv7 with regards to routing, see a more complete discussion in our GitHub repository.

2) Simulation Parameters: Other DTCP configuration options include the port of the DTCP receiving server, the desired probability for each of expected and unexpected disruptions, and the length of the timeframe. For the BPA, configurations include the default bundle lifetime and the set of actions the BPA should take when a status report notifies the agent that a bundle was lost. Finally, simulation parameters included minimum and maximum bundle sizes to send, simulation length, minimum and maximum sending frequency of bundles, and simulation identifier. The particular simulations analyzed in this report are contained within Section III.

D. *Other Design Decisions*

Several other design decisions were made. One such decision was using JSON to send bundles instead of Concise Binary Object Representation (CBOR) [31]. CBOR is a more dense and less human-readable replacement for JSON mentioned many times in the BPv7 specification. Implementing all of CBOR's specifications was out of the scope of this project. This potentially slows down some of our results, but should do so proportionally, allowing trends to still exist. This, however, makes our implementation not fully compliant with RFC 9171. See Section V-A4 for further discussion.

Another such decision was logging. Logging was implemented as an asynchronous thread, so while it should not have majorly impacted performance, it still slowed down the protocol some and is likely reflected in the results. However, data gathering is inherently important to the project, as is the ability to test different situations and see what errors may arise, so logging is pervasive in the program. An attempt was made to make tracking of the status of any bundle as clear as possible, so logging is done in every stage of a bundle's lifetime, including a unique identifier containing creation time, destination, etc.

III. Analysis

Individual graphs for each scenario are available in our GitHub repository.

A. *Scenarios and Network Statistics*

We prioritized measuring latency from sending application layer (A) to receiving application layer (B) as that is what matters most to the end user. Further discussion on additional network analysis is discussed in Section V-A1.

To measure the impact of various network situations, we created 8 scenarios labelled 0-8 in binary. For example, 010. The first bit represents low (0) or high (1) density of packets, where low density of packets is represented by packets only being sent every $200ms \pm 50ms$ and high density is a packet every $50ms \pm 5ms$. The second bit represents the size of the packets, with small (0) being $40B \pm 20B$ and large (1) being $10KB \pm 1KB$. The last bit represents low and high frequency of expected disruptions, with low (0) giving a disruption probability of 1% and high (1) a probability of 25%. Disruptions were checked every and lasted 100ms. The simulation results presented in this paper sent packets for 50 seconds from A to B and ran until B received every bundle and administrative overhead bundles stop.

B. *High Density Results*

Referring to Figure 1, what we observe is the delay from A to B over time in three tests with packets being sent very frequently (first bit). The results show that when large bundles are present in the network and being sent frequently (111/green), the network delay increased linearly with time while bundles were being sent. After bundles stopped being sent altogether, it took about 50 *more* seconds to finish sending the bundles.

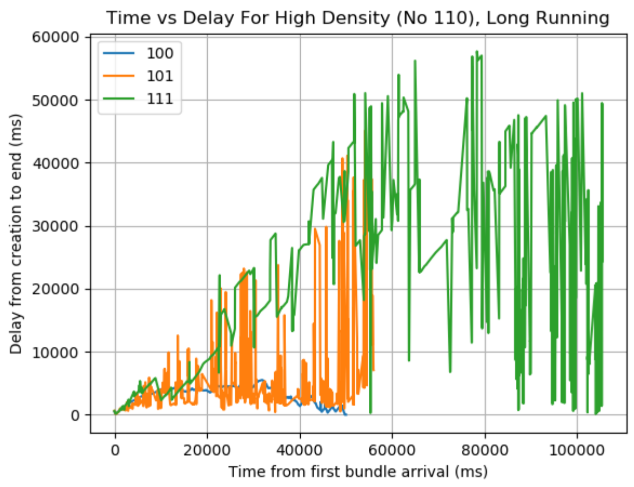

Fig. 1. Delay from application layer to application layer between sender and receiver, high density tests without the 110 test (which has high density of packets, high packet size, and low density of expected downs). Packets stopped being sent from the application layer at $t = 50s$.

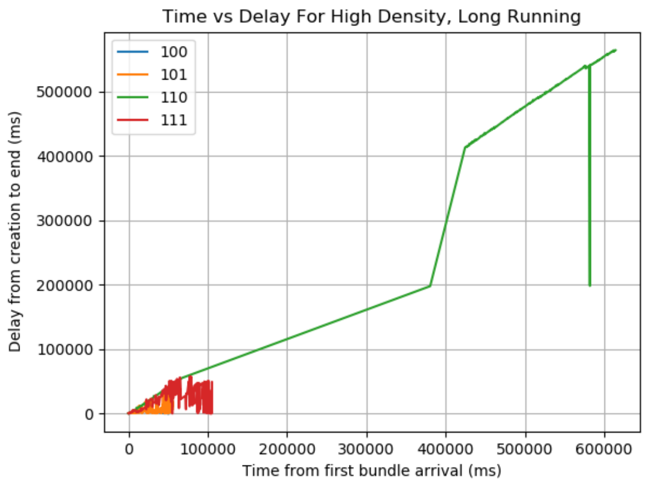

Fig. 2. Delay from application layer to application layer between sender and receiver, high density tests. Same as Figure 1 with the 110 test added. Packets stopped being sent from the application layer at $t = 50s$.

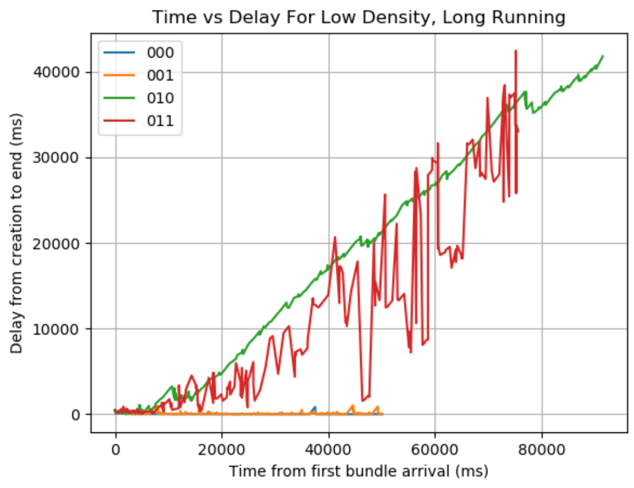

Fig. 3. Delay between application layers of sender and receiver, low density tests. Packets stopped sending from the application layer at $t = 50s$.

C. High Packet Density, Large Packets, Low Frequency of Interruptions

In Figure 2, the 110 case is of particular interest. When there was low frequency of interruptions, the nodes in the network had so much overhead in resending that at a certain point, new bundles started timing out before they were able to be sent. This caused Status Reports to be sent back to A, triggering a resend and causing thrashing in the network. After increasing the resend timeout increase from 1 second to 100 seconds (in several iterations, the last of which we consider the most extreme case we wanted to study), we saw that with a 100 second increase in TTL on each resend, *eventually* the bundle TTL was able to surpass the amount of time spent thrashing in the network. This happened around 420s, as seen in the graph, after which bundles started being delivered again (after the pause around 385s).

This is our most important result of yet as it shows that BP does not run well under what is considered standard networks loads in today's network, which includes very high density of packets that are not small. What BP was originally intended for, tolerating large packets with high-frequency and large delays (which the vast majority of modern networking protocols cannot handle at all), it meets its expectations. This is seen in the low density simulations in Figure 3.

IV. Architectural Improvements

RFC 9171 mentions in several locations that there is work needed in investigating parts of it, or leaves matters up to implementation. During the implementation phase of this project, we perceived a series of deficiencies in the current BP(v7) architecture based on the RFCs, that need to be addressed for potential future deployment.

A. Potential Architectural Flaw

Violation of Single Responsibility Principle (SRP): This is the most severe issue we raise in this paper. When an administrative record is received, either the BPA must parse it or the AA must de-serialize it. Therefore, the BPA needs an administrative record processing function or the AA needs a network serialization function; hence, receiving or responding to an administrative record violates the single-responsibility principle (SRP), part of the software engineering principle SOLID [32], as they are two separate layers within the bundle layer with conflicting responsibilities. Recall that the BPA is supposed to be the only layer that knows what a bundle is, and the AA is functioning partially as a secretary for the BPA, so it should have exclusive knowledge of what to do with administrative records and what they are (since they are content that gets wrapped in a bundle). Additionally, this issue leads to low cohesion and high coupling between the BPA and the AA, violating the software engineering principles of High Cohesion, Low Coupling, and Information Expert in another software engineering framework GRASP [33]. One way to remediate this is by placing the administrative element (AE) in the BPA layer instead of the AA layer, as those need to be coupled by the nature of what they do. This increases cohesion of the two parts (AA and BPA) individually while reducing their coupling by requiring the AE to be in the BPA layer.

B. Missing Specifications

Unspecified Convergence Layer: The convergence layer is largely left as an implementation detail in RFC 9171 and 5050. It is unclear what responsibilities are required for a convergence layer and how it should interact with BP. While it is true that convergence layers are defined in some RFCs, how convergence layers are implemented can drastically change the performance and functionality of the protocol, and the interfaces available to the application layer. As an overlay architecture, the importance of convergence layer and what BP needs from it must not be undermined. Moreover, RFC 9171 requires the convergence-layer protocol to provide congestion control, but such traffic-controlling features are closer to the responsibilities of the transport layer (or below). Some questions that could be addressed by 'BPv8' might include:

- How should the convergence layer receive packets? Is it required to hold onto partial packets for some amount of time until the rest can be filled in?
- What responsibilities should the CLA have that are explicitly *not* given to any other layer? For example, URI interpretation, routing, etc.
- Should the CLA be responsible for knowing anything about and/or handling (un)expected network disruptions?

Unspecified Subject and Layer for Acknowledgment and Reliable Transfer: The acknowledgment requested from the user application flag is not specific. It is unclear who (which layer, agent, etc.) is responsible for acknowledging, and there are a variety of things for which acknowledgments could be requested from this single flag. Also, delay-tolerant acknowledgments (perhaps in batches to reduce network load) could be desirable, even if it is much delayed due to batching or disruptions. We recommend including such protocol specifications. Moreover, per RFC 4838 [3], even if end-to-end acknowledgment is optional, DTNs are still responsible for providing hop-by-hop transfer of reliable delivery in some way (such as custody transfer [34]). In contrast, RFC 9171 does not explicitly require acknowledgments—or any sort of reliable transfer including custody transfer—for packets sent between two connected points in the network, seemingly violating the requirement of hop-by-hop reliable delivery responsibility by omission. RFC 9171 by itself, in effect, does not require BP to be reliable. Besides the potential performance issues (e.g., congestion, storage, etc.), it is unclear why custody transfer is left as optional, was removed in RFC 9171, or relies on other layers/protocols to handle it. See [34]–[36] for more information on the importance of custody transfer.

C. Missing Critical Features

Due to space constraints, we have reserved some of the more detailed criticisms that deal with potential deployment and implementation issues for our GitHub repository. Two big concerns covered there are issues like the *undefined notion of clock accuracy and method of synchronization* and *absence of identification of certain routing responsibilities*, which can both easily cause undesirable network patterns if no requirements are stated about them.

V. Conclusion

A framework that can be used to study BPv7 was introduced. Our implementation is lightweight; it contains only the required portions of RFC 9171 (except the CBOR data format) and is easier to understand (written in a popular programming language, JSON instead of CBOR, etc.) compared to other implementations available online. More importantly, our framework is easily configurable and extendable. Users can simulate more complicated network configurations and scenarios by modifying the Mininet configuration files as they desire. Our convergence layer (DTCP, Section II-A) then allows users to configurably simulate 'faulty' networks by generating disruptions of any pattern.

We also expect that our framework can be used for educational purposes, such as introducing and teaching students about the inner workings of BP and DTN, since our implementation is simpler than the others available online. Some parts of the implementation inherently are more difficult to write this way, especially while attempting to allow for maximum data collection via logging while measuring network speed, but for this reason, many parts were abstracted for the sake of clean external interfaces, with further investigation into its workings only required when studying that particular section.

The experimental results obtained (Section III) are based on simple experiments but give some insights into the network behavior of BP that could help guide design decisions in future iterations of BP. For example, as Section III-C shows, BPv7 may perform poorly under typical network loads.

We finally present a list of potential architectural, specification, and deployment-related flaws of BP, and suggest solutions or directions toward them in Section IV.

A. Future Works

There are several avenues for future research where our framework can be used. These are only a few such examples.

1) More Avenues of Analysis: Our current implementation studies end-to-end lifetime of a bundle, but analysis can and should be performed in the future on topics such as period spent in specific parts of the BPv7 ecosystem, and other behaviors. Much of the information for these analyses have been logged, but they are more complicated and may bring to light new considerations about BPv7.

2) Expanded Network Configurations: One such example is having more complicated network configurations, as currently testing is just done between two nodes communicating through a forwarding host, but no testing was performed involving several hosts sending to each other through a single or multiple forwarding hosts, and most communication other than status reports went in a single direction.

3) Data Corruption: BP is inherently designed with CRCs to be able to detect data corruption from transmission. This is not tested in our implementation, and no data corruption should occur in mininet, but artificial data corruption similar to how disruptions are handled in DTCP could allow for exploration of this topic.

4) Encoding: Our current implementation does not use CBOR [31], a more lightweight data encoding than JSON for sending over the network. RFC 9171 requires its use, but we went with JSON due to its clarity in implementation and the ease it provides in debugging processes such as log analysis, contributing to our goal of implementing BPv7 at the simplest level. However, this makes our implementation not fully compliant with RFC 9171. Viewing the effects of switching to this or some other efficient encoding from JSON may impact results, though trends about the network as a whole should be unaffected by a mere change in encoding because the relative size of bundles should remain approximately the same.

5) Metadata Optimization: In some cases, metadata can create many small bundles (like status reports) that can greatly burden the network. One potential avenue for improvement is the aggregation of status reports into a single bundle over a longer period. The status reports can add up to a significant amount of data when a large number of bundles are sent frequently. Shrinking them into a single bundle could alleviate this concern, and should be investigated.

References

[1] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '03. ACM, 2003, p. 27–34.

[2] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-tolerant networking: an approach to interplanetary internet," *IEEE Communications Magazine*, vol. 41, no. 6, 2003.

[3] L. Torgerson, S. C. Burleigh, H. Weiss, A. J. Hooke, K. Fall, D. V. G. Cerf, K. Scott, and R. C. Durst, "Delay-Tolerant Networking Architecture," RFC 4838, 04 2007.

[4] K. Scott and S. C. Burleigh, "Bundle Protocol Specification," RFC 5050, 11 2007.

[5] S. Burleigh, K. Fall, and E. J. Birrane, "Bundle Protocol Version 7," RFC 9171, 01 2022.

[6] A. Vasilakos, Y. Zhang, and T. Spyropoulos, Eds., *Delay Tolerant Networks: Protocols and Applications*. Boca Raton, FL: CRC press, 2016, wireless Networks and Mobile Communications Series.

[7] R. Pitts, K. Nichols, M. Holbrook, K. Gifford, A. Jenkins, and S. Kuzminsky, "Dtn implementation and utilization options on the international space station," in *SpaceOps 2010 Conference*, 2010.

[8] A. Schlesinger, B. M. Willman, L. Pitts, S. R. Davidson, and W. A. Pohlchuck, "Delay/disruption tolerant networking for the international space station (iss)," in *2017 IEEE Aerospace Conference*, 2017.

[9] A. Jenkins, S. Kuzminsky, K. K. Gifford, R. L. Pitts, and K. Nichols, "Delay/disruption-tolerant networking: Flight test results from the international space station," in *2010 IEEE Aerospace Conference*, 2010.

[10] R. Short, A. Hylton, R. Cardona, R. Green, G. Bainbridge, M. Moy, and J. Cleveland, "Towards sheaf theoretic analyses for delay tolerant networking," in *2021 IEEE Aerospace Conference*. Big Sky, MT: IEEE, 2021, pp. 1–9.

[11] A. Hylton, R. Short, R. Green, and M. Toksoz-Exley, "A mathematical analysis of an example delay tolerant network using the theory of sheaves," in *2020 IEEE Aerospace Conference*, Big Sky, MT, 2020.

[12] A. Hylton, J. Cleveland, R. Dudukovich, D. Iannicca, N. Kortas, B. LaFuente, J. Nowakowski, D. Raible, R. Short, B. Tomko, and A. Wroblewski, "New horizons for a practical and performance-optimized solar system internet," in *2022 IEEE Aerospace Conference*. IEEE, 2022.

[13] E. Koutsogiannis, S. Diamantopoulos, and V. Tsaoussidis, "A dtn testbed architecture," in *2009 International Conference on Ultra Modern Telecommunications & Workshops*. St. Petersburg, Russia: IEEE, 2009.

[14] G. Araniti, N. Bezirgiannidis, E. Birrane, I. Bisio, S. Burleigh, C. Caini, M. Feldmann, M. Marchese, J. Segui, and K. Suzuki, "Contact graph routing in dtn space networks: overview, enhancements and performance," *IEEE Communications Magazine*, vol. 53, no. 3, 2015.

[15] R. Dudukovich and D. E. Raible, "Transmission scheduling and routing algorithms for delay tolerant networks," in *34th AIAA international communications satellite systems conference*, ser. AIAA 2016-5753. Cleveland, OH: AIAA, Oct 2016, p. 5753.

[16] J. Deaton and B. Blanding, "Bpv6 and bpv7 coexistence in delay tolerant networking (dtn)," NASA Technical Reports Server (NTRS), Tech. Rep. 20210010630, 2021.

[17] A. Bisacchi, C. Caini, and S. Lanzoni, "Design and implementation of a bundle protocol unified api," in *2022 11th Advanced Satellite Multimedia Systems Conference and the 17th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, Graz, Austria, 2022.

[18] D. Ta, S. Booth, and R. Dudukovich, "Towards software-defined delay tolerant networks," *Network*, vol. 3, no. 1, pp. 15–38, 2023.

[19] C. Choudhari and D. Niture, "Disruption tolerant network (dtn) for space communication: An overview," in *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, 2022, pp. 1–5.

[20] A. Penning, L. Baumgärtner, J. Höchst, A. Sterz, M. Mezini, and B. Freisleben, "Dtn7: An open-source disruption-tolerant networking implementation of bundle protocol 7," in *International Conference on Ad-Hoc Networks and Wireless (AdHoc-Now 2019)*. Luxembourg, Luxembourg: Springer, 2019, pp. 196–209.

[21] L. Loiseau and N. Izvorski, "dtn7-kotlin: Delay-tolerant networking software suite for kotlin, bundle protocol version 7," https://github.com/NodleCode/dtn7-kotlin/, 2021.

[22] RightMesh, "Terra: Lightweight and extensible dtn library," https://github.com/RightMesh/Terra/, 2019.

[23] L. Baumgärtner, J. Höchst, and T. Meuser, "B-dtn7: Browser-based disruption-tolerant networking via bundle protocol 7," in *2019 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, Dec 2019, pp. 1–8.

[24] M. Demmer, A. McMahon, R. Martins, D. Long, and R. H. *et al.*, "Dtn2: Dtn reference implementation," https://github.com/delay-tolerant-networking/DTN2, 2005, last updated on Jun 27, 2016.

[25] S. Burleigh, "Interplanetary overlay network: An implementation of the dtn bundle protocol," in *2007 4th IEEE Consumer Communications and Networking Conference*. Las Vegas, NV: IEEE, 01 2007, pp. 222–226.

[26] J. Greifenberg and D. Kutscher, "Rdtn: An agile dtn research platform and bundle protocol agent," in *Wired/Wireless Internet Communications*. Springer Berlin Heidelberg, 2009, pp. 97–108.

[27] NASA, "High-rate delay tolerant network (hdtn) software," https://github.com/nasa/HDTN, 2019, last updated on May 2, 2023.

[28] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, no. 19. ACM, Oct 2010, pp. 1–6.

[29] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. ACM, Aug 2014, pp. 1–6.

[30] M. Nottingham, "URI Design and Ownership," RFC 8820, 06 2020.

[31] C. Bormann and P. E. Hoffman, "Concise Binary Object Representation (CBOR)," RFC 8949, 12 2020.

[32] R. C. Martin, M. Martin, and M. Martin, *Agile principles, patterns, and practices in C#*. Prentice Hall, 2007.

[33] C. Larman, *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. Pearson Education India, 2012.

[34] K. Fall, W. Hong, and S. Madden, "Custody transfer for reliable delivery in delay tolerant networks," Intel Research Berkeley, Tech. Rep. IRB-TR-03-030, 2003.

[35] F. Noviani, D. Stiawan, S. D. Siswanti, T. W. Septian, M. A. Riyadi, F. Aljaber, and R. Budiarto, "Analysis of custody transfer on moving bundle protocol of wireless router in delay tolerant network (dtn)," in *2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, 2017, pp. 50–53.

[36] K. Fall and S. Farrell, "Dtn: an architectural retrospective," *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 5, pp. 828–836, 2008.