Technical Paper
Journal of Aerospace System Engineering
Vol.18, No.4, pp.89-95(2024)

EISSN 2508-7150
http://dx.doi.org/10.20910/JASE.2024.18.4.89

통신 링크 데이터 온보드 저장 및 접근: 포도송이 연결리스트

구철회[1,†]

[1]한국항공우주연구원

Onboard Store and Access for Communication Link Data: Grape Linked-List

Cheol Hea Koo[1,†]

[1]Korea Aerospace Research Institute

Abstract

This paper introduces an effective and convenient method for utilizing onboard memory space to process remote commands, telemetry, and interplanetary network protocol data in satellite onboard systems. By enhancing the doubly linked list data structure to store and make accessible variable-length communication protocol data either sequentially or at variable locations, the paper enhances memory capacity utilization. The concept of 'grape' is introduced into the doubly linked list data structure to manage variable-length data and its access, with performance verification conducted through its reference implementation. This novel approach to linked lists is termed 'grape.'

초 록

본 논문은 원격명령과 텔레메트리 및 행성간 네트워크 프로토콜 데이터 처리가 위성 온보드에서 행해질 때 적용할 수 있는 효과적이고 용이한 온보드 메모리 공간 활용 방법을 소개한다. 이중 연결리스트 자료구조를 개선하여 고정이 아닌 가변 길이 통신 프로토콜 데이터를 순차적 또는 가변위치에 저장하고 접근 가능하도록 하는 기능을 제공함으로써 메모리 용량 대 활용성을 높일 수 있다. 가변 길이 데이터를 저장 및 액세스하기 위해서 이중 연결리스트 자료구조에 포도 송이 개념을 도입하고 이의 참조 구현을 통해 성능 검증을 수행하였으며, 이 새로운 개념의 연결리스트를 포도송이 연결리스트로 지칭하였다.

Key Words : Onboard memory(온보드 메모리), Interplanetary Network(행성간 네트워크), Doubly Linked-list(이중 연결리스트), Grape Linked-list(포도송이 연결리스트), Space link(우주링크), Delay-/Disruption-Tolerant Networking(결합허용네트워크)

1. Introduction

Each Protocol Data Unit (PDU) in a space communication transaction must be stored in onboard memory before processing. A memory handling function that only allows sequential and fixed-size length storage and access may increase the effort required for handling these PDUs, as well as the overhead of designing and implementing the memory storage system. The inability to erase certain parts of the stored data in the onboard memory handling system might severely impact the operational concept of PDU management, and may necessitate a conservative approach to handling abnormal conditions that are not predicted in normal operational scenarios. Random storage and access, as opposed to sequential methods, can be achieved using a doubly linked-list data structure, enabling operations such as read, write, copy, move, and

Received: Mar. 13, 2024 Revised: Jun. 07, 2024 Accepted: Jun. 26, 2024
† Corresponding Author
Tel: +82-42-860-2145, E-mail: chkoo@kari.re.kr

remove. However, using a doubly linked-list with dynamic memory allocation is suboptimal in highly constrained embedded software, such as spacecraft onboard systems. Most spacecraft systems prefer static, pre-allocated memory pool with fixed lengths to avoid the complexities of dynamic memory allocation and reconfiguration, as observed. An improved method of handling onboard memory holding PDUs may be employed during abnormal situations encountered during PDU processing. For example, an extreme case of resetting or clearing the onboard memory may be unnecessary for minor corruption in issued PDU blocks during transactions.

In this study, we introduce a modified handling concept for doubly linked-list data structure, referred to as the '*grape linked-list*,' aimed at reducing the need for extensive reconfiguration of onboard memory storage, such as resetting or clearing the entire memory, in response to minor PDU corruption due to harsh space environment[1].

We believe the novelty of the 'grape linked-list' arises in systems composed of various sizes of PDUs and plagued by frequent intermittent packet delivery issues. This is because the traditional doubly linked-list data structure demands more attention and processing power to manage such situations.

This paper is structured as follows: Section 2 presents a summary of the concerns regarding onboard memory processing for space communication link data. Section 3 introduces the new concept of Grape Linked-List as the main theme of this study. In Section 4, various use cases are presented to demonstrate the application of the core concept of the study and the proposed technique. Section 5 compares the results in terms of temporal and implementation complexities. Finally, Section 6 presents the study's conclusion.

2. Problems of Concern

In an onboard memory system that only allows sequential access to a static and pre-allocated fixed-length memory capacity, several circumstances may unnecessarily complicate the recovery function and process. This complication can limit the smooth and efficient usage of memory blocks onboard. It is essential to avoid this situation as much as possible because significant recovery actions are required, making the system difficult to manage and operate, either partially or entirely.

2.1 Invalidation of Processing Data

For any reason, partial or entire PDU blocks may become invalidated due to anomalies resulting in rejection, service denial, or no-operation. The affected PDU blocks must be cleared from onboard memory before processing or recovered through retransmission. In an onboard memory system that only offers sequential access service and limited PDU block manipulation capability, if this situation occurs, the only viable solution may be to clear the entire onboard memory. This system cannot distinguish between normal PDUs and faulty ones. However, this limitation can be resolved by selectively removing or handling the faulty PDU block from the current onboard memory block chain. No empty spaces are allowed to be left after deletion, resulting in dead spaces in an onboard memory system. This issue can be mitigated by using a doubly linked-list, which effectively restores those unnecessary blocks to the onboard memory system.

2.2 Processing of Out-of-order PDU

It is natural to receive bundles out-of-order through the IPN (Interplanetary Network) due to retransmission for missing blocks during transactions. In general, any incomplete PDU blocks should be temporarily stored in a designated temporary buffer. Within a highly congested communication system and in challenging network environments like space, this temporary memory buffer experiences frequent I/O operations, including insertion or deletion, to arrange the out-of-order delivered PDU blocks.

During transactions in IPN, each PDU block can be configured to have various lengths and may be aggregated with others for more efficient handling, such as by applying block channel coding. As mentioned earlier, problems may arise when out-of-order PDU blocks, which are not supposed to have their lengths known, arrive and need to be serialized.

In systems that prefer fixed memory block allocation and static pre-allocation over dynamic allocation, such as embedded computing systems in spacecraft, it is often necessary to concatenate multiple unit memory blocks to support larger PDU blocks. After using the concatenated memory block, it is then disassembled back into unit memory blocks and awaits the arrival of another set of PDU blocks.

3. Grape Linked-List

As stated above, we require a mechanism to properly address these limitations. Below is a summary of the key points to consider:

- The onboard memory management system should support functions for inserting, deleting, copying, and moving blocks. These functions shall not produce any residual side effects on the onboard memory system or the PDU processing operation.
- The onboard memory management system should be able to adapt to various lengths of incoming or outgoing PDU blocks using fixed-size memory blocks. The concatenated memory that has been used must be dismantled into its original small memory blocks so that these blocks can be utilized for other necessities.

In this section, we present novel enhancements to a doubly linked list to address the aforementioned considerations. We refer to it as the Grape Linked-List, *GLL*, and summarize the important definitions of this newly introduced linked list structure. This concept of GLL is inspired by the mechanism of doubly linked-list[2], unrolled linked list[2], and skip-list[3].

3.1 Theorems

In this subsection, we list up important theorems to GLL.

Theorem 1: GLL's baseline

GLL's baseline is a doubly linked list. Any aspect not explicitly addressed in GLL's theorems follows the definitions of a doubly linked list. Fig. 1 shows the typical configuration and operation of doubly linked-list.

Theorem 2: new member variables

GLL has three distinguished member variables to effectively conduct the necessitated functions.

- super previous: indicates access point to previous grape bunch
- super next: indicates access point to next grape bunch
- vertex: indicates access point to self

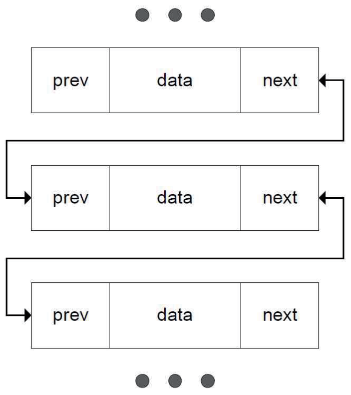

Fig. 1 Typical Doubly Linked-List

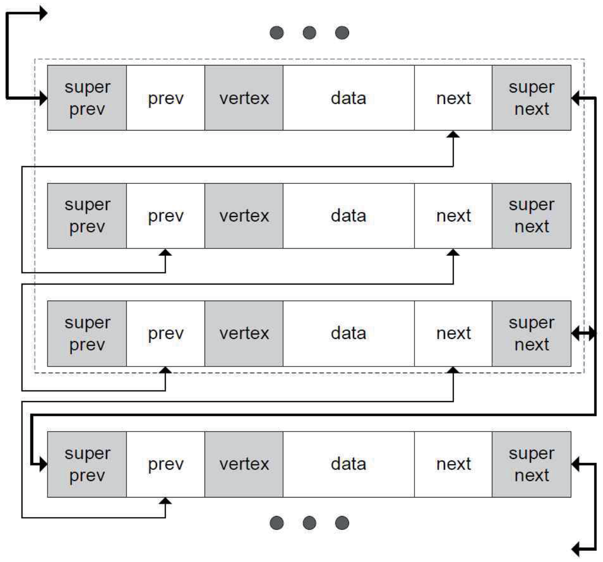

Fig. 2 Behavior of Grape Linked-List

Vertex in the GLL must have a valid address, indicating that it cannot be NULL, even when there's only a single item in the list of GLL. Vertex, newly introduced in this study, is one of the biggest differences from the last study[4]. It aims to reduce the required traversal steps for accessing adjacent GLL objects at the 'grape' level. Vertex is the unique value that indicates the object at the 'grape' level itself. Each unit list item at this level should have a single value for this throughout the 'grape' level.

Super previous and super next can have a valid address pointing to an object in the GLL if they exist; otherwise, they shall be NULL. For example, the super previous of the first object in a GLL should be NULL, and the super next of the last object in a GLL should be NULL. The super previous pointer in a non-vertex should be set to NULL to avoid overhead from updating variables during list changes. Similarly, the super next pointer in a non-vertex that is not the tail should be set to NULL for the same performance reasons. In other cases, the super previous and super next pointers should have a valid address to point to the existing super previous and next 'grape' level objects.

The 'Prev' and 'Next' pointers in the GLL adhere to the conventions of a typical doubly linked list. Fig. 2 shows structure and behavior of GLL.

Theorem 3: insertion and deletion

The insertion and deletion operations in GLL depend on whether the user wants to operate at the 'grape' level or not. The insertion and deletion operations can be performed at either the level of individual list elements or at the 'grape' level.

If the user wants to perform it at non 'grape' level, i.e., simply unit list item level, the function shall execute it using the currently targeted GLL object at the 'grape' level. If no GLL object exists, the insertion operation shall result in creating a new GLL object at the 'grape' level.

If a GLL object contains only one list item, deleting that single list item will also remove the GLL object simultaneously, regardless of the user's chosen deletion level.

In a GLL object, the size can be adjusted freely by inserting or deleting list items. These adjustments should not leave any empty spaces in the GLL object or onboard memory. Deleted items are restored to the memory buffer pool for reuse.

Theorem 4: traversing list items

The traversing operations, i.e., locating previous or next object of unit list item level or ‘grape’ level shall be provided.

The traversing operations at the unit list item level follow the same conventions as a typical doubly linked list.

The traversing operations at the ‘grape’ level are conducted using the ‘super previous’ and ‘super next’ methods to locate the vertex of the target GLL object. Fig. 3 shows the flow of traversal behavior among GLL objects.

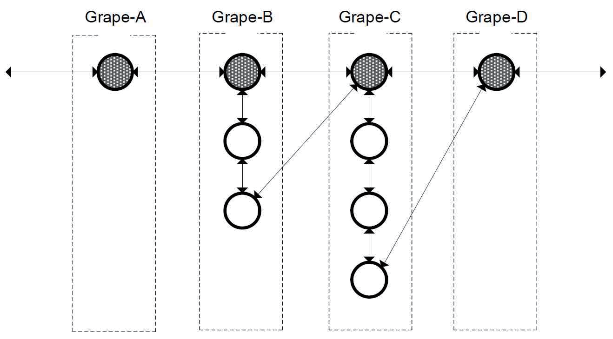

Fig. 3 Traversal of the Grape Linked-List

As a reference, it is important to note that efficient traversal into a 'grape' block and its elements within the onboard memory system, including GLL objects, is beneficial. The following variables in software implementation will be useful for expediting location to specified positions in linked-list items.

- list_head: Indicates the beginning of an onboard memory system or block.
- list_curr: Represents the current position of the list item being handled or referenced.
- list_tail: Marks the end of an onboard memory system or block. It indicates the last list item of the final 'grape' object in the onboard memory system. Other than the last 'grape' block, the last list item can be accessed by referring to the 'prev' position designated by the vertex’s 'super next' position as depicted in Fig. 3.

If list_curr indicates the middle of a 'grape' block, referencing the next 'grape' block requires accessing the vertex value to locate its ' super next' position, as only the top of the 'grape' block has valid references to 'super previous' or 'super next' positions as noted. It is essential practice to update these three variables whenever a change occurs at the 'grape' level or the unit list item level.

Theorem 5: copying and moving list items

The copying and moving operations adhere to the guidelines described in *Theorem 3* (insertion and deletion). When performing copying operations at the 'grape' level, a new 'grape' object is created in the targeted storage. When performing moving operations at the 'grape' level, the existing 'grape' object is being transferred to the targeted storage location. If these operations occur at the unit list item level, the list items being copied or moved will be positioned in another targeted storage either as a member of an existing 'grape' object or as a newly created 'grape' object, depending on the user's preference.

Theorem 6: merging

The merging operations in GLL objects require two parameters: the left-side 'grape' object and the right-side 'grape' object. The designated right-side 'grape' object is then attached to the designated left-side 'grape' object. The vertex of the designated right-side 'grape' object should be changed to match the value of the vertex in the designated left 'grape' object. Additionally, the super previous and super next in both 'grape' objects need to be updated accordingly.

4. Use Cases

Several use cases that may benefit from using the presented GLL in this paper are stated as follows.

4.1 Telecommand reception on onboard

Telecommands from the ground station cannot be assumed to be perfectly stored in onboard memory due to RF interference, solar conjunction, bad weather conditions, and other factors. Telecommands can be grouped into multiple command elements. When parts of the command block are missing during transmission, they can be safely stored in onboard memory temporarily, governed by the GLL as stated above, and await the missing parts through retransmission. Even when an unexpected length of commands is received for the missing parts, as it is not always possible to predict the incoming telecommand's volume, GLL certainly aids in completing the corrupted telecommand block and safely restoring it to the nominal command execution queue from the temporary buffer for further processing.

4.2 Telemetry handling on ground station

Usually, telemetry is composed of several groups that are logically related to specific functions or systems, such as thermal and electrical power, on a spacecraft.

The telemetry data needs to be stored along with a timestamp for future reference, particularly when diagnosing specific onboard times. If GLL is used for this purpose, insertion, traversal (search), or deletion operations are handled much more efficiently, as supported by the DBMS (Database Management System). Actually, DLLs are commonly used in the development of most DBMSs to implement indexes and data structures for efficient data management. GLL can enhance the performance of DLLs in data management tasks, even when only some parts of telemetry are received successfully.

GLL can support grouping of groups, making it possible to handle telemetry groups of spacecraft systems across subsystems for storing, searching, removing, and sorting.

4.3 Bundle handling in IPN

Several studies foresee that IPN (Interplanetary Network) in deep space will experience more corrupted bundles than conventional space communication due to the harsher communication environment it encounters[1]. The retransmission of lost or corrupt bundles is the baseline operational concept in IPN. Therefore, uncompleted segments of bundles need to be properly handled rather than simply rejecting or denying service to the delivered bundles.

If parts of bundles in a segment of the IPN are lost or corrupted during transmission, all received bundles need to be stored in a temporary buffer in a convergence layer such as LTP (Licklider Transmission Protocol)[5]. Each bundle segment can be characterized by its length, priority, and demanded reliability. The lost or corrupted bundles are requested for retransmission; when they arrive at the receiver side without errors, they need to be merged with the already stored bundle segments in the correct order, ensuring orderly delivery. These operations and processes can benefit from GLL due to the frequent insertion, deletion, copying, and moving operations required to complete the corrupted segment.

Thanks to GLL, the onboard memory system does not require a partitioned area to store multiple segments of IPN bundles for processing, whether nominal or temporary. Instead, GLL offers flexible and adaptive methods for handling various applications in space, including telemetry, telecommand, and IPN bundle transactions.

5. Performance Analysis

This section presents some thoughts on the complexities of temporal dynamics and implementation challenges for GLL. In this section, we present a comparative analysis of the Big-O notation for the GLL method against the traditional doubly linked-list. By evaluating the time and software complexities of various operations, we aim to highlight the efficiency and potential advantages of our new approach.

5.1 Temporal Complexities

Table 1 shows the comparison results of the temporal complexities between GLL and traditional doubly linked-list.

As shown in Table 1, GLL may offer improved temporal performance compared to DLL, especially for storage composed of large logical blocks in various processing streams, such as insertion and deletion. GLL can significantly save processing time during sorting because fewer item-moving operations are needed when larger logical blocks are used, as the contents of the logical blocks are highly diverse and do not require sorting. This indicates that GLL is an efficient way to handle PDU blocks for telecommand, telemetry, or any space network protocol, such as DTN(Delay-/Disruption-Tolerant Networking).

Table 1 Temporal Complexity Comparison with DLL and GLL

Function	DLL	GLL
Creation	O(1)	O(1)
Insertion*	O(n)	O(log n)
Deletion**	O(n)	O(log n)
Search***	O(n)	O(log n)
Move/Copy	O(n)	O(n) *or* O(log n)****

*insertion to an arbitrary position
**deletion to a logical block
***can be utilized to sorting operation as well
****fewer condition checking is needed for a logical block

Table 2 Software Complexity Comparison with DLL and GLL

Test Function	Statements	% Branches	Max Complexity	Avg Depth
Ref*	83	28.9	26	3.22
GLL**	233	26.6	71	4.44
△(%)***	280.72	92.04	273.08	137.89

*Original reference function of linked-list memory block allocation not supporting GLL
**Revised function of linked-list memory block allocation to support GLL
***Difference between the reference code and GLL code, calculated by (GLL/Ref) × 100 (%)

5.2 Implementation Complexities

The software complexity of onboard memory block handling functions increases when compared to the original version, which only supports doubly linked-lists. An analysis using SourceMonitor[6], available as open source, reveals that the maximum complexity of the most sophisticated function in GLL, specifically linked-list block allocation within the source code of KARI DTN software[7], exhibits approximately a 173% increase in maximum complexity and a 180% increase in code length compared to the original source code, which only supports doubly linked-lists, as shown in Table 2.

However, due to the minimization of loop elements in both the concept and implementation of GLL, it is anticipated that there will be no performance issues, and the use of GLL is expected to enable efficient handling of space link data.

6. Conclusion

The operation of satellite missions and the transaction of IPN bundles can benefit from this GLL technique and concept. The occurrence of chronic deletion or complete memory area clearing in fixed-length memory systems with only sequential access, caused by simple or partially corrupt data, is nearly eliminated in an onboard memory system using GLL.

Each 'grape' block can be efficiently traversed by accessing its vertex values. Since every element of a 'grape' block possesses a valid vertex value, accessing the previous or next block can be nearly instantaneous. GLL are being applied to internally develop KARI DTN software. This results in source code that is simpler and more concise compared to the older version of the KARI DTN software, which only uses a doubly linked list for the above-mentioned bundle handling issues.

This study emphasizes that GLL is more efficient in handling cases where received PDUs are partially broken and require repair via retransmission, and where PDUs are not delivered in order, allowing for out-of-order storage due to retransmission. GLL can efficiently handle and repair intermittently broken PDUs or packets received non-sequentially in space communication networks.

However, GLL requires additional memory to store information about super next and super previous nodes, as well as vertex addresses, compared to the traditional DLL approach. Additionally, implementing the GLL concept requires more attention due to scattered checking points during pointer handling.

For further study, the GLL implementation code can be integrated into an existing DLL implementation rather than developed as an independent codebase. Since most of GLL's functions utilize DLL functions, an abstraction model of GLL over DLL can be created. This model will enable developers to easily grasp the concept of GLL. The complexity of GLL software, as mentioned earlier, makes it challenging to implement directly on top of a DLL codebase.

Acknowledgment

This work was funded by the Korea Meteorological Administration's Research and Development Program "The Development of Meteorological Satellite Operation and Application Technology" under Grant (KMA2013-03720).

References

[1] A. G. Voyiatzis, "A Survey of Delay- and Disruption-Tolerant Networking Applications", *Journal of Internet Engineering*, vol. 5, no. 1, pp. 331-344, 2012.

[2] N. Karumanchi, Data structures and algorithms made easy, CareerMonk Publications, Bombay India, 2010

[3] C. P. Rangan, Handbook of Data Structures and Applications, Ch14, 2nd Ed., CRC press, FL U.S., 2018

[4] C. H. Koo, S. Y. Kang, H. Y. Choung, J. H. Lee, S. C. Lee, and M. S. Lee, "Improvement Study for Satellite CSA Function for Easy Maintenance of Mission Commands," *Proc. Of the KSAS 2023 Fall Conference*, Hongcheon, Korea, pp. 708-709, November 2023.

[5] M. Ramadas, S. C. Burleigh, and S. Farrell, "RFC 5326, Licklider Transmission Protocol Specification," IRTF DTN Research Group, 2008. [Online] Available: https://tools.ietf.org/ html/rfc5326

[6] https://www.derpaul.net/SourceMonitor/

[7] C. H. Koo and S. C. Burleigh, "Structural Considerations for Generating and Handling LTP Report Segments from an Interoperability Testing," *Journal of Korean Institute of Communications and Information Sciences*, vol 47, no. 12, pp. 2065-2077, 2022.