Journal of
Sensor and Actuator Networks

MDPI

Article

Parallel Eclipse-Aware Routing on FPGA for SpaceWire-Based OBC in LEO Satellite Networks

Jin Hyung Park [1], Heoncheol Lee [1,2,*] and Myonghun Han [3]

[1] School of Electronic Engineering, Kumoh National Institute of Technology, Gyeongbuk 39177, Republic of Korea
[2] Department of IT Convergence Engineering, Kumoh National Institute of Technology, Gyeongbuk 39177, Republic of Korea
[3] Agency for Defense Development, Daejeon 34186, Republic of Korea
* Correspondence: hclee@kumoh.ac.kr; Tel.: +82-54-478-7476

Abstract

Low Earth orbit (LEO) satellite networks deliver superior real-time performance and responsiveness compared to conventional satellite networks, despite technical and economic challenges such as high deployment costs and operational complexity. Nevertheless, rapid topology changes and severe energy constraints of LEO satellites make real-time routing a persistent challenge. In this paper, we employ field-programmable gate arrays (FPGAs) to overcome the resource limitations of on-board computers (OBCs) and to manage energy consumption effectively using the Eclipse-Aware Routing (EAR) algorithm, and we implement the K-Shortest Paths (KSP) algorithm directly on the FPGA. Our method first generates multiple routes from the source to the destination using KSP, then selects the optimal path based on energy consumption rate, eclipse duration, and estimated transmission load as evaluated by EAR. In large-scale LEO networks, the computational burden of KSP grows substantially as connectivity data become more voluminous and complex. To enhance performance, we accelerate complex computations in the programmable logic (PL) via pipelining and design a collaborative architecture between the processing system (PS) and PL, achieving approximately a 3.83× speedup compared to a PS-only implementation. We validate the feasibility of the proposed approach by successfully performing remote routing-table updates on the SpaceWire-based SpaceWire Brick MK4 network system.

Keywords: LEO network; K-Shortest Paths; Eclipse-Aware Routing; FPGA; SpaceWire

check for updates

Academic Editor: Xiang Chen

Received: 26 May 2025
Revised: 29 June 2025
Accepted: 9 July 2025
Published: 15 July 2025

Citation: Park, J.H.; Lee, H.; Han, M. Parallel Eclipse-Aware Routing on FPGA for SpaceWire-Based OBC in LEO Satellite Networks. *J. Sens. Actuator Netw.* **2025**, *14*, 73. https://doi.org/10.3390/jsan14040073

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the increasing demand for networks with reliability and speed around the world, satellite-based networks are attracting attention as a complementary solution to traditional communication infrastructure. In particular, LEO satellite networks, which can effectively cover the world and provide critical low latency in the network, are attracting attention as key technologies for next-generation satellite-based networks to replace existing satellite-based networks. LEO satellite networks provide Internet services with low latency even in areas where traditional communication infrastructure is scarce. In addition, large-scale satellite constellation can connect the world into one network, and the development of satellite communication technology promotes high-speed data transmission through satellite-based networks. Taking advantage of these advantages, LEO satellites can be utilized in various fields such as telecommunication, agriculture, environmental surveillance, and emergency response [1]. Moreover, recent military-oriented studies reaffirm that the space segment

constitutes the pivotal backbone for global communication reliability and security [2]. The growing interest in satellite-based networks has led to the need for global satellite communication standards. Built to meet these demands is the SpaceWire protocol. SpaceWire is designed for real-time data transmission between space-mounted systems and built on the low-voltage differential signaling (LVDS) scheme, which consumes less power, and most satellites are operational with SpaceWire as their communication protocol [3]. In LEO spacecraft, SpaceWire serves as the on-board backbone bus, interconnecting sensors, the on-board computer (OBC), mass-memory units, and RF subsystems via low-power LVDS links that employ wormhole routing. Because each router exposes only a fixed-format routing table, a mechanism capable of dynamically managing and updating these tables is essential.

Routing algorithms are indispensable in LEO satellite networks. However, the unique characteristics of low Earth orbit (LEO) make it impractical to apply conventional routing protocols. Compared to medium Earth orbit (MEO) and geostationary Earth orbit (GEO) satellites, LEO satellites operate at lower altitudes, which reduces latency and path loss but at the same time introduces significant challenges [4]. Because each LEO satellite's coverage area is much smaller than that of MEO or GEO systems, global coverage requires the deployment of hundreds or even thousands of satellites. Moreover, the rapid movement of LEO satellites generates a dynamic topology that is fundamentally different from terrestrial and traditional satellite networks, posing one of the major challenges for LEO constellations [5]. This fast mobility leads to frequent changes in inter-satellite connections and demands new routing strategies capable of managing these variations efficiently. Although real-time routing computations have been necessary in existing satellite systems, the complexity of routing in LEO networks places even greater demands on the real-time processing capabilities of on-board computers (OBCs). Consequently, extensive research efforts have been devoted to developing novel routing strategies specifically tailored for LEO satellite networks [6–8].

The strategic management of a satellite's energy resources is equally critical in LEO satellite networks. When a satellite is exposed to sunlight, its solar panels generate power to support on-board operations such as communications and data processing, and the excess energy is stored in on-board batteries. However, when the satellite enters Earth's shadow (eclipse), it can no longer generate power and must rely solely on the stored battery energy. During this eclipse period, the charge–discharge cycles of the battery are constrained, and the depth of discharge (DoD) measures the fraction of stored energy that has been consumed. As DoD increases, battery degradation accelerates and overall battery lifetime is reduced. Therefore, to extend battery life during eclipse periods, it is essential to minimize excessive energy consumption and keep DoD as low as possible [9–11].

The conventional OBC has faced significant challenges in real-time routing and data processing due to its limited hardware resources and computational performance. To address these issues, this study proposes the use of an FPGA—offering superior computation capabilities and abundant logic resources—as the next-generation OBC [12]. We implemented on the FPGA both the Eclipse-Aware Routing (EAR) algorithm, which accounts for satellite energy consumption rate, eclipse duration, and expected transmission load, and the K-Shortest Paths (KSP) algorithm, which concurrently identifies K optimal paths to flexibly accommodate the frequent link disruptions inherent in LEO constellations. While prior work has demonstrated EAR and KSP acceleration via GPU-based parallelization, our approach adopts a space-qualified FPGA platform with a PS-PL heterogeneous architecture [13]. Recent comparative studies report that FPGAs deliver 1.2–22.3 × lower energy-per-frame than state-of-the-art GPUs in embedded-vision pipelines and therefore recommend FPGAs for power-constrained platforms [14,15]. Although several GPU-

accelerated or dynamically routed schemes have been proposed for LEO networks, their high-power envelopes and limited radiation tolerance render them impractical for on-board deployment. Consequently, we adopt an FPGA accelerator to achieve efficiency while meeting space-grade reliability requirements. Specifically, the programmable logic (PL) region employs parallel processing and pipelining techniques to dramatically reduce latency, thereby meeting the stringent real-time requirements of LEO satellite networks.

Finally, we conducted experiments by integrating the routing algorithms with the SpaceWire Brick MK4—an IEEE 1355-standard SpaceWire device for space communication networks [16]. These experiments included updating the Brick's routing table based on the FPGA-executed KSP algorithm results, thereby demonstrating the networking feasibility in a realistic satellite environment. The main contributions of this paper are as follows:

1. We verified the feasibility of implementing a K-Shortest Paths (KSP) algorithm, tailored for LEO satellites, on an FPGA and secured real-time performance by reducing computation time through acceleration and parallelization; additionally, we incorporated the Eclipse-Aware Routing (EAR) algorithm, which considers eclipse conditions and energy consumption rates.
2. We demonstrated the practical applicability of our approach by integrating the FPGA with SpaceWire-based systems and implementing dynamic routing-table updates on actual SpaceWire network hardware, SpaceWire Brick MK4.

The remainder of this paper is organized as follows. Section 2 describes the topology characteristics of LEO satellite networks, their energy-efficiency challenges, and the real-time routing limitations of conventional OBCs. Section 3 presents the Eclipse-Aware Routing (EAR) algorithm, the K-Shortest Paths (KSP) algorithm, and the methods used for parallelization and acceleration on FPGA. Section 4 details the experimental setup and discusses the results. Finally, Section 5 summarizes the findings and outlines future work.

2. Problem Description

2.1. Characteristics of LEO Network Topologies

As mentioned earlier, LEO satellite networks have a different dynamic topology than ground networks and existing satellite networks. LEO satellites operate at an altitude between 500 KM and 2000 KM from Earth, so they orbit the Earth at a very high speed and perform continuous positional movement. This means that, over time, the connection between satellites and the connection to the ground changes, which means that links between specific satellites can only be maintained for a short period of time, and the creation and release of links occur repeatedly. As a result, routing technologies used in existing static networks cannot be directly applied, and routing technologies that can reflect the changing connection status in real time are needed.

In this study, LEO satellite network routing is performed using the KSP algorithm. Existing routing algorithms calculate a single optimal path, which is cumbersome to re-operate when the connectivity information between satellites changes. The KSP algorithm generates K optimal path candidates to generate candidate groups and goes through the process of selecting the path with the least cost among them. This is an algorithm that can ensure the stability of the LEO satellite network because, even if the connectivity information between satellites is changed, only other optimal paths need to be selected from the candidate group [17,18]. A typical method for implementing the KSP algorithm is Yen's algorithm and it can be implemented with many other algorithms. In this paper, the K-Shortest Path problem is solved with a Depth First Search (DFS)-based approach. We selected DFS because, although Yen's algorithm can be mapped to an FPGA, its spur-path expansion produces many data-dependent branches and priority-queue updates that break a one-clock-cycle initiation interval (II = 1), making efficient pipelining and thus real-time

routing impossible. DFS, by contrast, traverses one branch at a time with a fixed-depth stack, so every loop can be fully unrolled, scheduled at II = 1, and implemented with minimal control logic.

2.2. Energy Efficiency Problem in LEO Networks

Efficient energy management is critical for LEO satellites, which orbit the Earth every 90 min and may experience eclipses lasting several tens of minutes per revolution [19]. During these periods, satellites rely entirely on stored power, and a higher depth-of-discharge (DoD) accelerates battery degradation, thereby shortening both battery and satellite lifespans. Traditional routing protocols designed to minimize path length or latency do not account for such battery-intensive conditions. When applied to LEO networks, they can unevenly increase DoD on certain satellites, leading to imbalanced energy consumption and potential in-flight outages [20]. To address this, we propose an Eclipse-Aware Routing (EAR) algorithm that distributes traffic evenly by considering each satellite's energy consumption rate, eclipse duration, and expected transmission load. This approach balances DoD levels across the constellation, maximizes energy efficiency, and extends operational lifespan, ultimately reducing the substantial operating costs of LEO satellite networks [20]. Moreover, previous studies leveraged GPUs to parallelize and accelerate the EAR and KSP algorithms to achieve high performance; however, a key disadvantage of GPUs is that their high power consumption can actually reduce overall energy efficiency [15].

2.3. Problem of Real-Time Routing on OBCs with Limited Resources

The LEO satellite network requires real-time routing because of its rapidly changing link state and high mobility. However, the OBC mounted on the existing LEO satellite has limited computational power and memory [17]. This may not fully reflect the real-time performance that requires many and fast operations. To address this, we used an FPGA and verified the possibility of performing EAR and KSP algorithms in the PS area of the FPGA. After that, the PS-PL collaboration structure was implemented and the real-time routing possibility was increased by reducing the computational time through pipelining in PL.

3. Proposed Method

3.1. Overall Structure

The overall structural diagram of the algorithm used to verify the FPGA implementability of the EAR, KSP algorithm before pipelining is shown in Figure 1. Satellite two-line element (TLE) data is extracted by random time to generate connectivity data that connects nodes according to certain criteria. At this time, one node refers to one LEO satellite. After that, this data is designated as an input value to the KSP to perform the KSP algorithm. In the KSP algorithm, the DFS function is called to find all the destination paths that can be found. The DFS method searches for the end of one node through depth expansion and then, if there are no more nodes that can be expanded, all the paths that can reach the destination through backtracking back to the previous node will be searched. After all path searches are completed, the DFS function will pass all the paths that can reach the destination node from the departure node to the KSP function, and in the KSP function, the paths received are sorted in ascending order based on cost and the K paths with the lowest cost are returned. Finally, the optimal path is selected by considering the satellite's energy consumption rate, eclipse duration, and expected transmission load in the EAR algorithm.

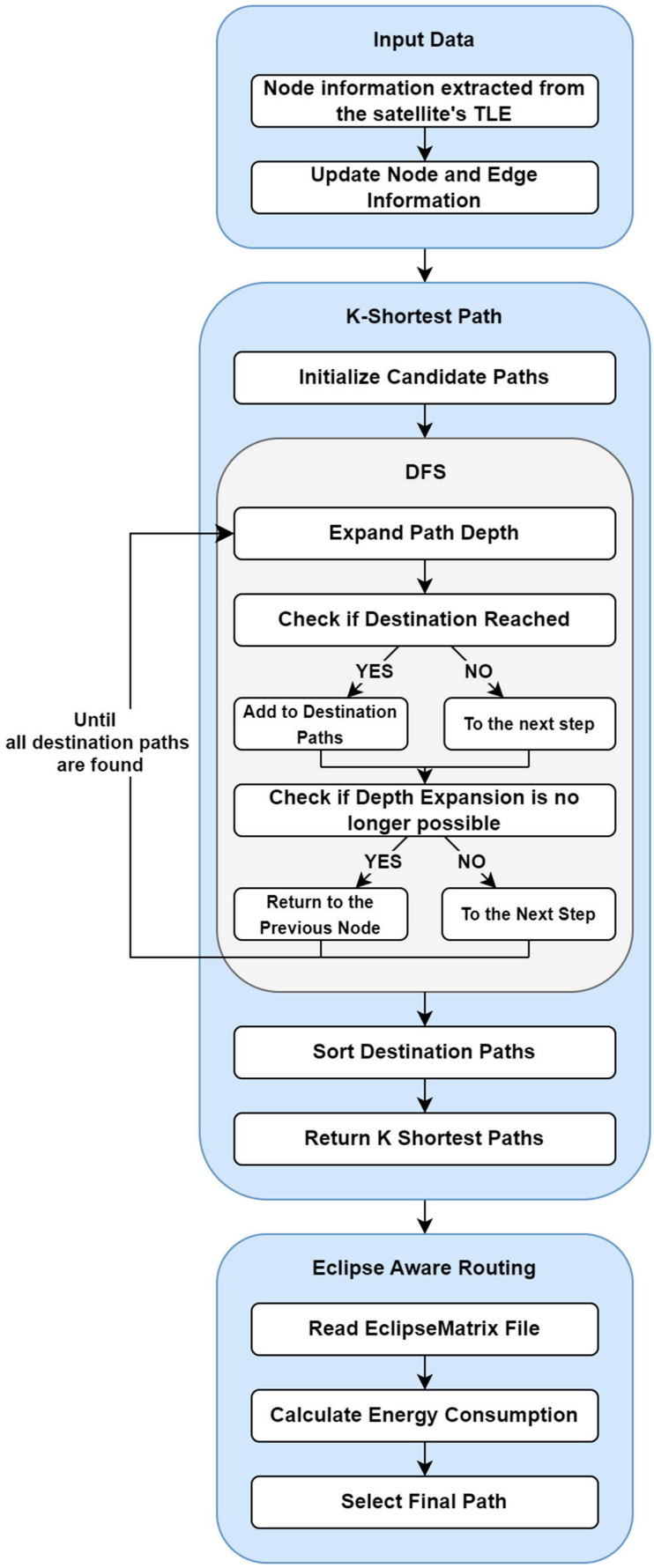


Figure 1. KSP and EAR Algorithms for FPGA Feasibility Verification.

3.2. Description for Eclipse-Aware Routing (EAR)

Algorithm 1 is Pseudo-code representing the Eclipse-Aware Routing method. The goal of this algorithm is to select a stable data transmission path while reducing energy consumption in the LEO satellite network. After performing the KSP algorithm, this algorithm takes K paths and compares the various conditions to select the optimal path. First, it initializes the energy consumption value for the basic work. Then it receives K paths and checks whether the satellites included in each path enter the eclipse zone. If a satellite exists in a solar eclipse zone, it calculates the duration of the eclipse and estimates the expected amount of communication during that period and the energy value used at that time. Next, the total energy consumption is calculated by summing the estimated energy value and the basic operating energy. Although the DoD is not explicitly calculated, it considers the risk of battery consumption for satellites with high traffic volume while staying in the eclipse zone and considers it indirectly. When evaluating a path, it calculates the maximum energy consumption of the path based on the satellite that consumes the most energy within each path and finally selects the path with the lowest value. This can prevent excessive load from being concentrated on a specific satellite and reducing battery efficiency and life. This approach is similar to previous studies predicting DoD through eclipse time and energy concentration without directly computing DoD [9,11].

The equation used in the EAR algorithm to estimate energy consumption during an eclipse period is as follows.

$$Energy = A \times B + C \times (D + 1) \quad (1)$$

where the variables are defined as follows:

A: power consumption rate per unit time;
B: duration of the satellite eclipse;
C: energy required for a single data transmission or reception;
D: number of transmissions/receptions performed during the eclipse;

3.3. DFS-Based K-Shortest Path Search Algorithm

Algorithm 2 is the pseudo-code representing the DFS-based K-Shortest Path algorithm. The goal of this algorithm is to search through DFS for all routes that can be reached from the starting node to the destination node and then select the top K routes with the lowest cost. First, initialize the array required for the discovery. Set the allPaths array to store paths, the allCosts array to store path costs, and the visited array to record whether or not to visit. After verifying that the current node is the destination node, DFS extends the current path to each unvisited neighbor and adds the corresponding edge cost to the allCosts. When the destination node is reached during the discovery, it stores the path and cost to this day. Then, the search to another path is continued through backtracking, which allows all possible paths to be discovered. When the discovery is complete, the saved paths are sorted in ascending order of cost and the top K paths are stored in the paths array to return. DFS, implemented through its iterative structure and stack, facilitates the implementation of pipelining.

3.4. IP Design for Eclipse-Aware Routing on FPGA with Parallelization Achieved Through Pipelining

In this study, it was important to use both PS and PL to reduce the calculation time to ensure real-time performance. For this purpose, it is very important to reduce the calculation time using pipelining in the PL. For pipelining, the KSP algorithm was implemented in the PS area of the FPGA to measure which function takes the longest calculation time. Figure 2 is a table showing the results. Total execution time (KSP) is the total execution time of the KSP function, total execution time (DFS) is the total execution time for the DFS

algorithm to be performed within the KSP algorithm, neighbor expansion time is the time taken for the path to extend, backtracking time is the time to backtrack, pop & visit check time is the time to check whether the path has reached its destination, and sort & K-paths selection time is the time to sort and K stored results by reaching the destination from the DFS function. According to the above results, it was decided to run the KSP algorithm in a PL including the most time-consuming DFS algorithm and the sorting and selection of K results and pipelining some of the processes. To enable pipelining, the KSP algorithm was first converted into an intellectual property (IP) core, which is a reusable, standalone hardware block that can be instantiated within the FPGA fabric and interfaced through standard AXI buses. Figure 3 depicts the structural layout of the KSP and EAR algorithms executed across the processing-system (PS) and programmable-logic (PL) regions. Although its basic operational flow mirrors the PS-side algorithm diagram introduced in Figure 1, the KSP algorithm has been encapsulated as a dedicated IP core to run in the PL. Within this IP, the color-highlighted sections denote regions where pipelining has been applied to shorten execution time. Figure 4 presents a schematic overview of the pipelined KSP algorithm operating in the PL. Unlike the original implementation—whose computations were performed sequentially as in Order 1—the accelerated version leverages pipelining to launch a new operation every clock cycle, enabling parallel execution and dramatically reducing overall runtime.

Algorithm 1 Eclipse-Aware Routing

Input	eclipseMatrix[time][nodes] paths[]: result of KSP sampleTime NoE, TxE, RxE nTimeStep
Output	minPath

```
Initialize Constants: NoE, TxE, RxE
Initialize energyTable[pathCount][maxLength] = 0
For each pathIdx from 0 to topPath − 1;
    currentPath <= Paths[pathIdx];
    For satIdx from 0 to length(currentPath) − 1;
      currentSat <= currentPath[satIdx]
       If eclipseMatrix[timeIndex][currentSat] == 1;
          (startIdx, endIdx) <= locateEclipseInterval(eclipseMatrix,
          timeIndex, currentSat)
         eclipseDuration = (endIdx − startIdx) × sampleTime;
txEstimate <= round(eclipseDuration × random(0.01, 0.1));
Compute Energy Consumption:
baseEnergy <= eclipseDuration × NoE;
commEnergy <= (txEstimate + 1) × (TxE + RxE)
energyTable[pathIdx][satIdx] <= baseEnergy + commEnergy
End if
End for
End for

Compute Maximum Energy Consumtion per Path:
For each pathIndex from 0 to topPath −1;
maxEnergyPerPath[pathIdx] <= maximum value in
energyTable[pathIdx]);
End for
Select Optimal Path:
minPath <= index of Minimum value in maxEnergyPerPath
Return minPath;
```

Algorithm 2 DFS-based K-Shortest Path Algorithm

Input	numEdges[], startVertex, endVertex, K edges[MAX EDGES] ⟵ list of (src, dest, weight)
Output	paths[]

```
1.  Initialize allPaths[300][MAX VERTICES] <= 0
2.  Initialize allCosts[300] <= 0, pathCount <= 0, maxDepth <= 0
3.  stack <= (startVertex, depth = 0, cost = 0)
4.  visited[MAX VERTICES] <= false
5.  while stack not empty do
6.        (current, depth, cost) <= stack.pop()
7.  if visited[current] then continue
8.  visited[current] <= true
9.  path[depth] <= current
10. maxDepth <= max(maxDepth, depth)
11. if current == endVertex then
12. allPaths[pathCount] <= path[0.depth]
13. allCosts[pathCount] <= cost
14. pathCount <= pathCount + 1
15. visited[current] <= false
16. continue
17. end if
18. hasChild <= false
19. for each (src, dest, weight) in edges do
20. if src == current and not visited[dest] then
21. stack.push(dest, depth + 1, cost + weight)
22. hasChild <= true
23. end if
24. end for
25. if not hasChild then
26. visited[current] <= false
27. end if
28. end while
29. sort allpaths in allcosts ascending order
30. Save the top K paths from allPaths and their costs from allCosts into
31. paths
32. return paths
```

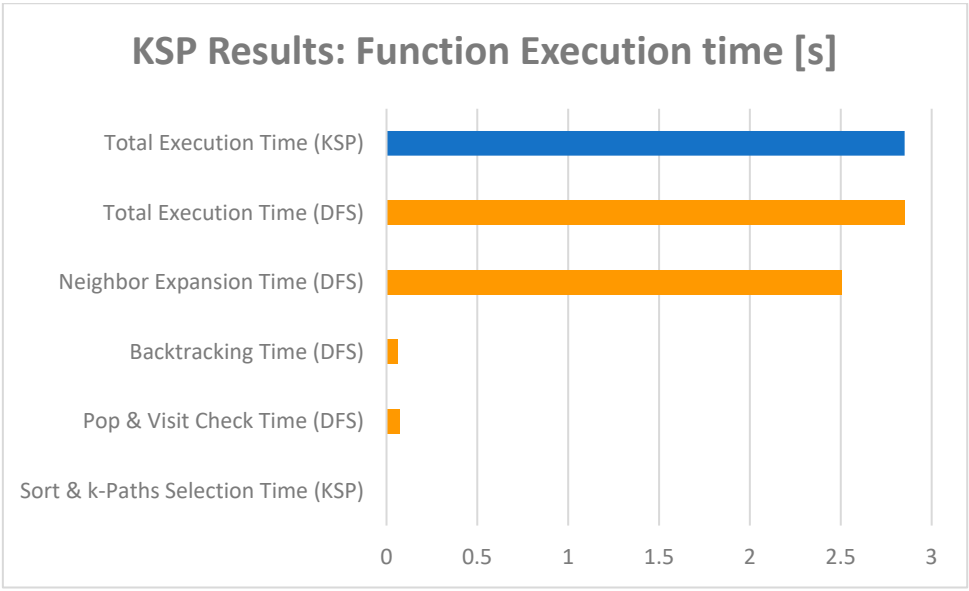


Figure 2. PS area profiling results of existing KSP algorithm.

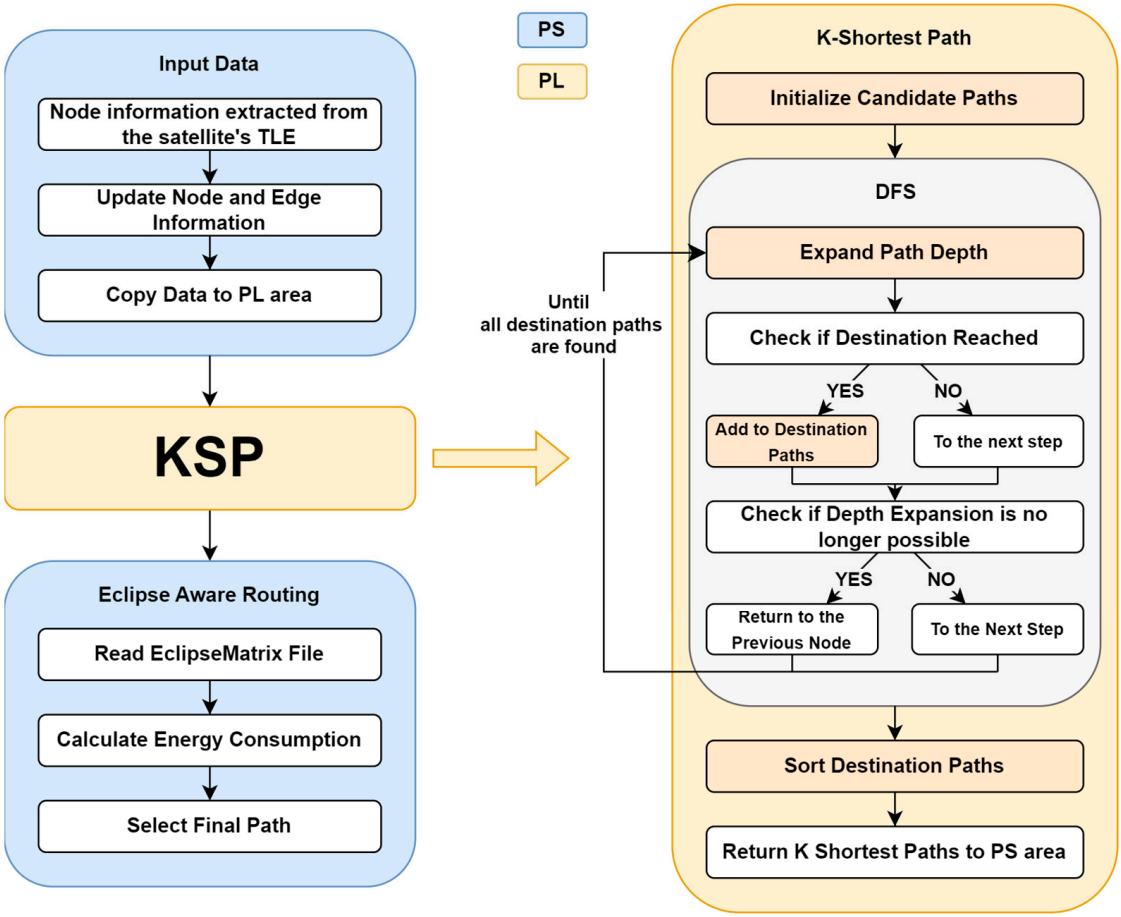


Figure 3. KSP and EAR Algorithms Running in PS and PL area.

CLK	1	2	3	4	5	6	7	8
Order 1	Set Candidate Paths	Expand Paths	Check Destination	Back Tracking	Set Candidate Paths	Expand Paths	Check Destination	Back Tracking
Order 2		Set Candidate Paths	Expand Paths	Check Destination	Back Tracking	Set Candidate Paths	Expand Paths	Check Destination
Order 3			Set Candidate Paths	Expand Paths	Check Destination	Back Tracking	Set Candidate Paths	Expand Paths
Order 4				Set Candidate Paths	Expand Paths	Check Destination	Back Tracking	Set Candidate Paths
Order 5					Set Candidate Paths	Expand Paths	Check Destination	Back Tracking

Figure 4. Schematically illustrated pipelined KSP algorithm.

4. Experimental Results

4.1. Experimental Setups

This experiment verifies satellite network routing using FPGA-accelerated KSP and EAR algorithms. Figure 5 presents the overall test architecture, and Figure 6 shows the real equipment. The testbed comprises two PCs, a single FPGA board, and two SpaceWire Brick MK4 units—hereafter MK4#1 and MK4#2. The host PC programs the FPGA over JTAG; the FPGA then executes the KSP and EAR kernels and returns the best route to the PC. Using this result, the PC remotely updates the routing table of MK4#2 across the SpaceWire network. MK4#1 and MK4#2 were controlled from the PC through the SpaceWire API.

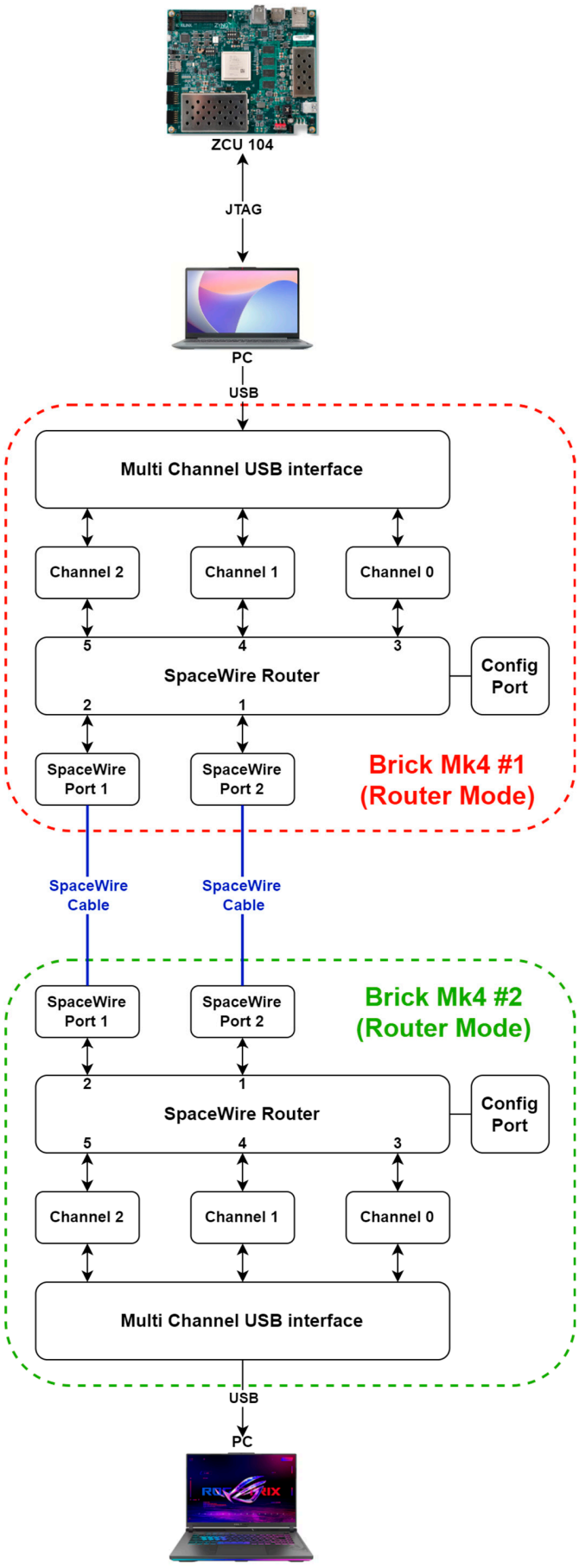


Figure 5. Structure for experimental environments.

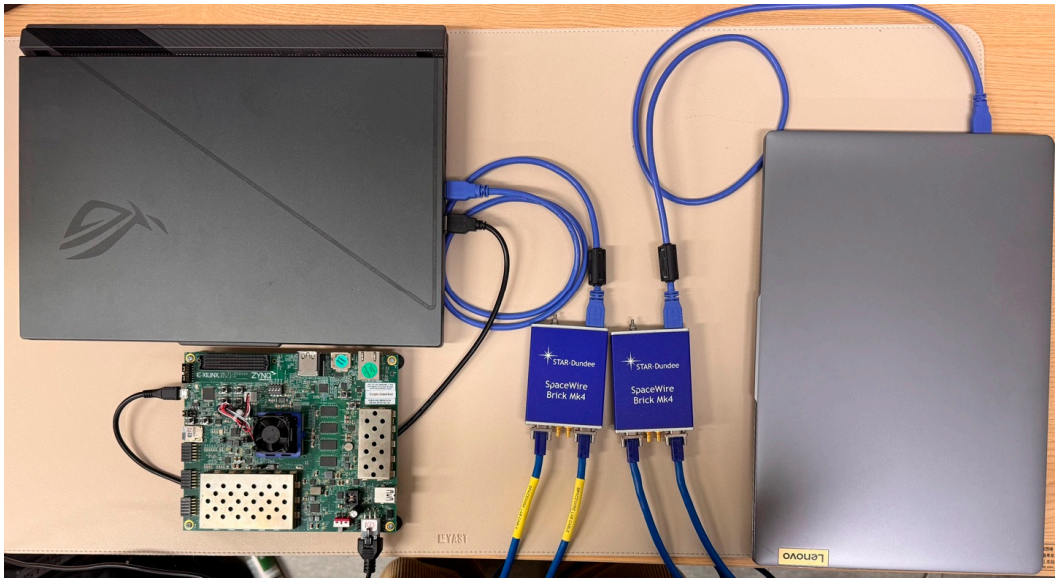


Figure 6. Configured real experimental environment.

In Figure 5 the two MK4 routers are interconnected with a SpaceWire cable. The Brick MK4 integrates a multi-channel USB 3.0 interface that conveys data, power, and control signals between the host PC and the router. Through this interface, Channel 0 serves as a dedicated control link for reading and writing the routing table and link parameters, while Channels 1 and 2 provide independent data paths for SpaceWire packets, thereby preventing contention during multi-link experiments. At the core sits a non-blocking SpaceWire router capable of path and logical addressing at up to 300 Mbit/s per link. Router configuration and status monitoring are accessed via a dedicated config port, and external connectivity is supplied by SpaceWire ports 1 and 2, which physically interconnect MK4#1 and MK4#2 through a SpaceWire cable that forms the actual data path between the two devices.

Figure 6 is a photograph of the laboratory setup used for the experiment. The two PCs are interconnected via SpaceWire equipment to the MK4 routers. The PC on the left is linked to the FPGA; after receiving the computation results from the FPGA, it remotely updates the routing tables and thus corresponds to the upper-side PC in Figure 5. The PC on the right, which checks whether the routing table has been correctly modified, serves as the lower-side PC in Figure 5.

The FPGA used in this experiment was UltraScale+™ MPSoC ZCU104. Because a central contribution of this work is an FPGA implementation that operates in concert with real SpaceWire hardware, a detailed description of the experimental platform is essential to demonstrate both the practicality and reproducibility of our approach. Since PS and PL are installed and we have enough resources for the current experimental setup, pipelining in PL was possible. The PC connected to the remote device MK4#2 verified the accuracy of the routing table's modification with the SpaceWire GUI. In this experiment, the performance of the PC was not an important factor in the experimental results, so the specifications of both PCs were omitted.

Table 1 ummarizes the key specifications of the FPGA used in this experiment. The Zynq UltraScale+™ MPSoC ZCU104 board is built around the Xilinx Zynq UltraScale+ MPSoC (ZU7EV) and offers four Cortex A53, two Cortex R5, Mali 400 MP2 GPUs with clocks up to 600 MHz, 2 GB DDR4, and a variety of high-speed I/O (PCIe, USB 3.0, SATA, etc.). This makes it an optimal environment for embedded FPGA acceleration and PS-PL collaboration and is suitable for KSP and EAR algorithm acceleration assessments such as large-scale graph

search. In this experiment, we evaluated the performance of the KSP and EAR algorithms based on the FPGA using resources from the UltraScale+™ MPSoC ZCU104.

Table 1. AMD Zynq UltraScale+™ MPSoC ZCU104 board major specification.

Category	Specification
SOC	Zunq UltraScale+™ MPSoc ZU7EV
Programmable-logic resources	504 K system logic cells, 461 K CLB flip-flops, 6.2 Mb distributed RAM, 11 Mb block RAM, 27 Mb UltraRAM, 1728 DSP slices
On-board memory	2 GB DDR4 component memory
Power consumption	15 W

4.2. LEO Network Topologies for Experiments

In this experiment, TLE data predicted at a future epoch for satellites passing over a designated observation point were collected, and only those satellites that met predefined selection criteria were retained for analysis. A total of 150 selected satellites participated in the communication experiment, and the K value was set to 15. Subsequently, all satellites located within a specified communication radius were connected by edges of equal cost (1), and each satellite was assigned a sequential identifier for analysis. This procedure produced a graph-based satellite network topology representing connectivity information.

4.3. K-Shortest Paths and Eclipse-Aware Routing Results

Table 2 compares the standalone KSP results with those obtained after EAR is applied. When routing from node 38 to node 98, KSP alone generates three candidate paths and selects path 1, whereas adding EAR changes the selection. EAR first identifies satellites that are in eclipse and computes a node cost for each one based on energy use, remaining eclipse time, and DoD. For each candidate path, the largest node cost is taken as the path metric, and the path with the smallest metric becomes the final choice. In this case, path 3 is selected after EAR is applied. In Table 2 nodes in eclipse are shown in red, each node's cost appears in parentheses, and the path metric for each route is highlighted in blue. This procedure optimizes the energy balance of the LEO constellation and mitigates battery degradation.

Table 2. KSP Algorithm and EAR Algorithm Results.

K paths	Paths	Cost
1	38 ⟶ 39 ⟶ 40 ⟶ 63 ⟶ 64 ⟶ 87 (7980) ⟶ 98 (7300)	6
2	38 ⟶ 39 ⟶ 40 ⟶ 63 ⟶ 74 ⟶ 75 ⟶ 86 ⟶ 87 (7980) ⟶ 98 (7300)	8
3	38 ⟶ 37 ⟶ 114 ⟶ 103 ⟶ 102 ⟶ 101 ⟶ 100 (7300) ⟶ 99 (7300) ⟶ 98 (7300)	8

4.4. Comparison Results Between Only PS and PS + PL

Table 3 compares the execution times of a PS-only configuration with those of a combined PS + PL implementation. As the edge data grows, the number of nodes the KSP algorithm must explore increases, because KSP expands all possible paths the computational load rises sharply, lengthening the runtime. Execution time also varies with the source node, since the complexity of the connectivity graph changes; accordingly, in the TLE snapshot used here, the graph rooted at source 2 (with destination fixed at node 98) is denser than the graph rooted at source 38. Across all four test cases, the PS + PL design consistently outperformed the PS-only baseline, achieving up to a 3.81× speed-up. These results indicate that the pipelined PL implementation was effective and demonstrate that an FPGA with pipelining is highly advantageous for executing the KSP and EAR algorithms on large graph workloads.

Table 3. Comparison KSP Algorithm results between only PS and PS + PL.

	PS (ms)	PS + PL (ms)
160 edge data/Departure 38/Destination 98	8.893	2.621
160 edge data/Departure 2/Destination 98	71.098	19.090
300 edge data/Departure 38/Destination 98	226.378	59.304
300 edge data/Departure 2/Destination 98	2450.603	638.854

4.5. Results with SpaceWire-Based Devices

In this study, we utilize SpaceWire, a high-speed serial communication standard established by the European Space Agency (ESA). We verify the dynamic routing performance of the proposed algorithm through the STAR Dundee's USB-based SpaceWire interface device, Brick MK4. First, we construct a network topology based on the satellite locations predicted from the artificially generated TLE. After finding K paths from origin to destination through the KSP algorithm, the EAR algorithm identifies the satellites located in the solar eclipse zone from K paths, calculates the expected energy consumption of each satellite, and compares it with the highest energy consumption of each path to select the path with the lowest energy consumption. Then, the RMAP function of Brick MK4 updates the routing table of BrickMK4, which is not directly connected to the PC connected to the FPGA, by sending the command to update the routing table generated by the PC to the SpaceWire network. Experimental results show that the proposed technique successfully explores the path that is optimal and minimizes energy consumption, unlike the path chosen by the traditional KSP algorithm. This shows that it provides significant benefits for energy balance optimization, battery degradation prevention, and optimal path selection in the LEO satellite network. Figure 7 illustrates the internal architecture of the Brick MK4 and the experimental scenario. In the test case defined in Figure 5 (source = node 38, destination = node 98), a standalone KSP search would select node 39 (channel 2) and route through port 5, whereas the EAR-enhanced algorithm is designed to choose node 37 (channel 1) and route through port 4. Figure 8 is a screenshot from the PC attached to the receive-side Brick MK4; the routing table was remotely updated by the FPGA-host PC so that the link was redirected to node 37, exactly as prescribed by the EAR result. These observations confirm that the proposed algorithm executes correctly on the FPGA and that the Brick MK4 routing table can be updated in real time over the network.

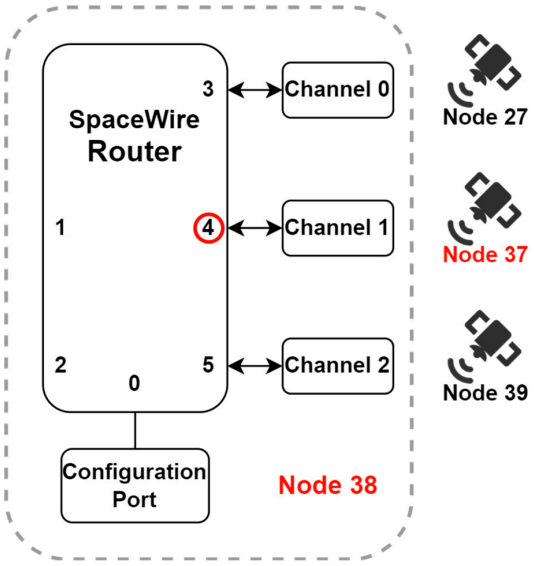


Figure 7. SpaceWire Brick MK4 inner structure and experimental scenarios.

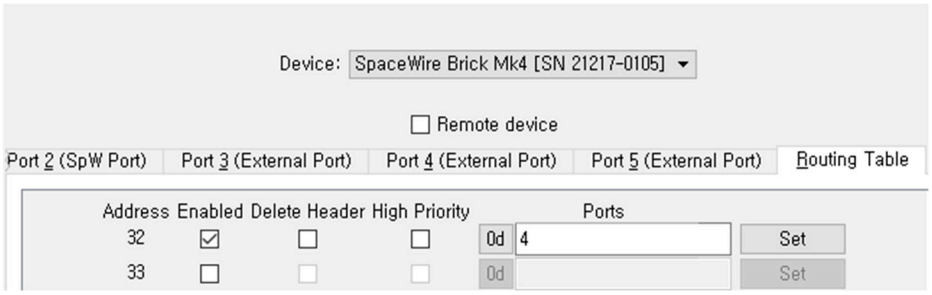


Figure 8. SpaceWire Device Configuration GUI screen viewed from the receiver's PC.

5. Conclusions

This paper proposed an FPGA-based parallel EAR method for LEO satellite networks, which achieves a 3.83× execution-time reduction over the conventional CPU-based method. We employed an FPGA to overcome the resource limitations of OBCs and to manage energy consumption effectively using the EAR algorithm. The proposed method first generates multiple routes from the source to the destination using KSP, then selects the optimal path based on energy consumption rate and eclipse duration, and estimates transmission load as evaluated by EAR. Also, the proposed method was parallelized with pipelining methods to reduce the computation time. Finally, the applicability of the proposed method to real space systems was validated by performing remote routing-table updates on SpaceWire Brick MK4 devices.

Author Contributions: Conceptualization, H.L. and M.H.; methodology and software, J.H.P.; validation, formal analysis and investigation, H.L. and M.H.; writing—original draft preparation, J.H.P.; writing—review and editing, H.L. and M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Agency for Defense Development, funded by the Korean government (Defense Acquisition Program Administration) (UI237032TG).

Data Availability Statement: The datasets presented in this article are not readily available because the data are part of an ongoing study.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Leyva-Mayorga, I.; Soret, B.; Röper, M.; Wübben, D.; Matthiesen, B.; Dekorsy, A.; Popovski, P. LEO Small-Satellite Constellations for 5G and Beyond-5G Communications. *IEEE Access* **2020**, *8*, 184955–184964. [CrossRef]
2. Zhang, Y.; Gao, X.; Yuan, H.; Yang, K.; Kang, J.; Wang, P.; Niyato, D. Joint UAV trajectory and power allocation with hybrid FSO/RF for secure space–air–ground communications. *IEEE Internet Things J.* **2024**, *11*, 31407–31421. [CrossRef]
3. Parkes, S.M.; Armbruster, P. SpaceWire: A spacecraft onboard network for real-time communications. *IEEE Trans. Nucl. Sci.* **2005**, *52*, 2277–2283.
4. Rani, S.; Malhotra, S. Comparative Study Of LEO, MEO & GEO Satellites. *Int. J. Res. (IJR)* **2014**, *1*, 1181–1186.
5. Wang, J.; Li, L.; Zhou, M. Topological dynamics characterization for LEO satellite networks. *Comput. Netw.* **2007**, *51*, 43–53. [CrossRef]
6. Liu, B.; Wang, L. Resource allocation algorithm for space-based LEO satellite network based on satellite association. *KSII Trans. Internet Inf. Syst.* **2024**, *18*, 1638–1658.
7. Zheng, X.; Liu, J.; Li, J.; Lv, D.; Liao, J.; Chen, X.; Hsu, T. A routing strategy for GEO/LEO satellite network based on dynamic delay prediction and link control. In *Proceedings of the Communications and Networking—17th EAI International Conference, ChinaCom 2022, Xi'an, China, 19–20 November 2022*; Gao, F., Wu, J., Li, Y., Gao, H., Eds.; Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering; Springer: Berlin, Germany, 2023; Volume 500, pp. 3–17.
8. Guo, Z.; Zhao, Y.; Li, J.; Sun, L. Load-balancing routing for LEO satellite network with distributed hops-based back-pressure. *Sensors* **2023**, *23*, 9789.

9. Macambira, R.D.N.M.; Carvalho, C.B.; de Rezende, J.F. Energy-efficient routing in LEO satellite networks for extending satellites' lifetime. *Comput. Commun.* **2022**, *195*, 463–475. [CrossRef]
10. Suchida, H.; Kawamoto, Y.; Kato, N.; Kaneko, K.; Tani, S.; Uchida, S.; Aruga, H. Efficient Power Control for Satellite-Borne Batteries Using Q-Learning in Low-Earth-Orbit Satellite Constellations. *IEEE Wirel. Commun. Lett.* **2020**, *9*, 809–812. [CrossRef]
11. Hussein, M.; Jakllari, G.; Paillassa, B. On Routing for Extending Satellite Service Life in LEO Satellite Networks. In Proceedings of the Globecom 2014—Symposium on Selected Areas in Communications: GC14 SAC Satellite & Space Communication, Austin, TX, USA, 8–12 December 2014; pp. 2832–2837.
12. Kim, D.; Lee, H.; Won, D.; Han, M. FPGA-based Inference Parallelization for Onboard RL-based Routing in Dynamic LEO Satellite Networks. *Int. J. Aeronaut. Space Sci.* **2024**, *25*, 1135–1145. [CrossRef]
13. Kim, H.; Lee, H.; Han, M. GPU-Accelerated Eclipse-Aware Routing for SpaceWire-Based OBC in Low-Earth-Orbit Satellite Networks. *Aerospace* **2025**, *12*, 422. [CrossRef]
14. Qasaimeh, M.; Denolf, K.; Lo, J.; Vissers, K.; Zambreno, J.; Jones, P.H. Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels. *arXiv* **2019**, arXiv:1906.11879.
15. Vaithianathan, M.; Patil, M.; Ng, S.F.; Udkar, S. Comparative study of FPGA and GPU for high-performance computing and AI. *ESP Int. J. Adv. Comput. Technol.* **2023**, *1*, 37–46.
16. STAR-Dundee Ltd. *SpaceWire Brick Mk4 Datasheet*; STAR-Dundee Ltd.: Dundee, UK, 2023.
17. Vissicchio, S.; Handley, M. Reliable Low-Delay Routing In Space with Routing-Oblivious LEO Satellites. *arXiv* **2024**, arXiv:2401.11490.
18. Bhosale, V.; Saeed, A.; Bhardwaj, K.; Gavrilovska, A. A Characterization of Route Variability in LEO Satellite Networks. In Proceedings of the Passive and Active Measurement Conference (PAM 2023), Virtual Event, 21–23 March 2023; pp. 313–342.
19. Sumanth, R.M. Computation of Eclipse Time for Low-Earth Orbiting Small Satellites. *Int. J. Aviat. Aeronaut. Aerosp.* **2019**, *6*, 15.
20. Tsuchida, H.; Kawamoto, Y.; Kato, N.; Kaneko, K.; Tani, S.; Hangai, M.; Aruga, H. Improvement of Battery Lifetime Based on Communication Resource Control in Low-Earth-Orbit Satellite Constellations. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 1388–1398. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.