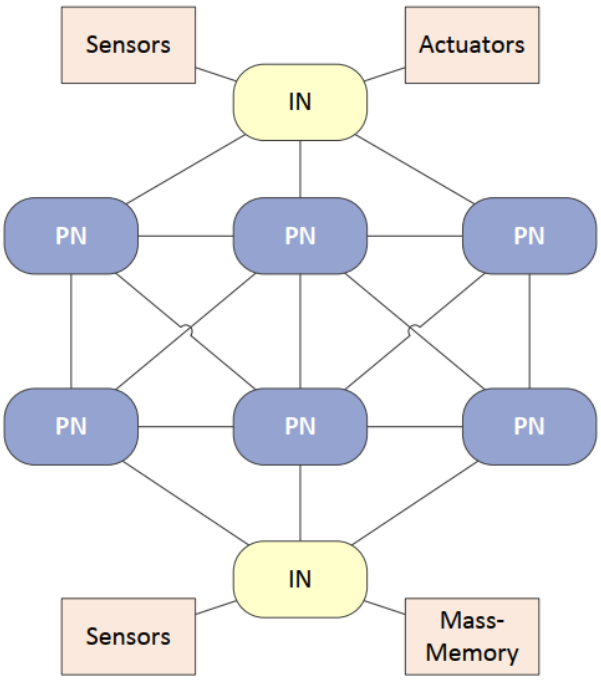
LULEÅ UNIVERSITY OF TECHNOLOGY

DLR

Master Thesis

Design and Analysis of a Dynamic SpaceWire Routing Protocol for Reconfigurable and Distributed On-Board Computing Systems



Prem Kumar Hari Krishnan

Master in Space Science and Technology

SpaceMaster, 2017-19

Abstract

Future spacecrafts will require more computational and processing power to keep up with the growing demand in requirements and complexity. ScOSA is the next generation on-board computer developed by the German Aerospace Centre (DLR). The main motivation behind ScOSA is to replace the conventional on-board computer with distributed and reconfigurable computing nodes which provides higher performance, reliability, availability and stability by using a combination of the COTS components and reliable computing processors that are space qualified. In the current ScOSA system reconfiguration and routing of data between nodes are based on a static decision graph. SpaceWire protocol is used to communicate between nodes to provide reliability. The focus of the thesis is to design and implement a dynamic routing protocol for ScOSA which can be used in future for not only communicating between the nodes but also for reconfiguration.

SpaceWire IPC is a customized protocol developed by DLR to provide communication between the nodes in a distributed network and to support monitoring, management and reconfiguration services. The dynamic routing protocol proposed in this thesis is primarily derived from the monitoring mechanism used in the SpaceWire IPC. PULL type monitoring mechanism is modelled and simulated using OMNeT++. The results obtained provide a qualitative outlook of the dynamic routing protocol implemented.

Acknowledgement

I would like to begin by thanking my supervisor Mr. Arnau Prat i Sala for his immense support and guidance throughout the thesis. I am also very grateful to the Deutsche Zentrum für Luft- und Raumfahrt (German Aerospace Center) for providing me this wonderful opportunity to work at their premises and learn more about the current state of the art technologies in the space domain.

Finally I take this opportunity to express my sincere gratitude and thank Dr. Anita Enmark for introducing me to the interesting topic of On Board Computers and agreeing to supervise my thesis.

Contents

List of Figures

List of Tables

Glossary

ACK	Acknowledgement
ACS	Attitude Control System
ATON	Autonomous Terrain-based Optical Navigation
BIRD	Bi-spectral Infrared Detection Satellite
CAN	Controller Area Network
CCSDS	Consultative Committee for Space Data Systems
COTS	Commercial off the shelf components
CPU	Central Processing Unit
DLR	Deutsche Zentrum für Luft- und Raumfahrt (German Aerospace Center)
DM	Dependable Multiprocessor developed by NASA
ECSS	European Cooperation for Space Standardization
EDAC	Error Detection and Correction
EOP	End Of Packet
ESA	European Space Agency
FDIR	Fault Detection Isolation and Recovery
FIFO	First In First Out Queue
FPGA	Field Programmable Gate Arrays
GDC	Gemini Digital Computer
GUI	Graphical User Interface
HB	Heart Beat
HiL	Hardware in Loop Setup
HPN	High Performance Node
IBM	International Business Machines
IEEE	Institute of Electrical and Electronics Engineers
IFN	Interface Node
IP	Intellectual Property
IPC	Inter Process Communication using SpaceWire
ISO	International Organisation for Standardization
JAXA	Japan Aerospace Exploration Agency
JWST	James Webb Space telescope

LVDS	Low Voltage Differential Signalling
NASA	National Aeronautics and Space Administration
NoC	Network on Chips
OBC	On Board Computer
OBC-NG	On Board Computer: Next Generation, predecessor to ScOSA
OBSW	On Board Software
OEM	Original Equipment Manufacturer
OMNeT++	Open source Discrete Event Simulation Framework based on C++
OSI	Open Systems Interconnection Model
OSPF	Open Shortest Path First Algorithm
PMU	Payload Monitoring Unit
RCN	Reliable Computing Node
RIP	Routing Information Protocol
RKA	Roscosmos State Corporation for Space Activities
RMAP	Remote Memory Access Protocol
RTC	Remote Terminal Controller
ScOSA	Scalable On-Board Computing for Space Avionics
SEU	Single Event Upset
SMT	Satisfiability Modulo Theory
SoC	System on Chip
SPARC	Scalable Processor Architecture
TRL	Technology Readiness Level
USB	Universal Serial Bus
WANET	Wireless Ad Hoc Networks

Chapter 1

Introduction

There has been a rapid growth in the evolution of OBC starting from the GDC developed by IBM which was used by NASA's Project Gemini. The GDC Gemini Digital computer was the first OBC used in a spacecraft (1964). It used a magnetic core memory with $7\,\mathrm{khz}$ clock weighing about $27\,\mathrm{kg}$. However 55 years later today the current generation of OBC's have evolved to a SoC architecture which essentially includes CPU, memory and input/output ports in a single chip[1]. The main purpose of the OBC is to provide computational performance for the spacecraft ranging from vehicle control, platform data processing, communication, etc. In orbit they have to withstand the electromagnetic conditions and radiation from the van Allen Belt. The OBC must be robust enough to vibrations and shock during stage separations and pyrotechnic releases. Finally OBC should fulfill the functional safety with redundancies in case of failures. Irrespective of the evolution, these implications still significantly influence the design of OBC[1].

1.1 Mission Statement and Purpose

Over the the past 55 years, fault-tolerant space systems have been used successfully because of their reliable architecture and software development processes with verification and validation. However due to the recent advancements in sensor technology and imaging systems, the size and complexity of space missions have grown rapidly. Space missions are usually very unique and mission specific, the complexity may be due to the high amount of payload data that needs to sent from the spacecraft, for example the JWST James webb Space telescope[2] or the Philae Lander[3] due to its autonomy in deep space. The trend in increase of OBSW onboard software size can be clearly seen in Figure 1.1

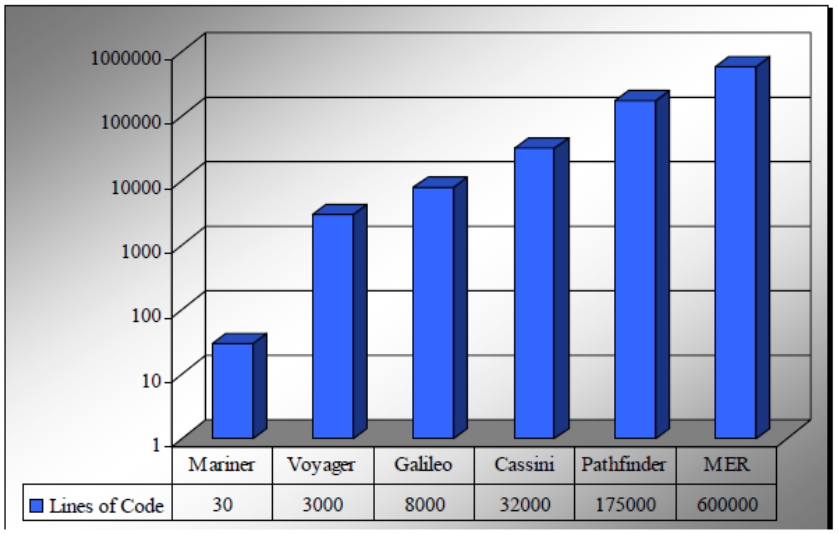


Figure 1.1: Size of software in spacecraft missions[4]

The current generation of fault tolerant processors such as LEON4FT[5] have the state of the art SPARC architecture with EDAC capabilities which provides high reliability, a platform to implement FDIR algorithms and is also radiation hardened. However they are very expensive and have limited processing power when compared to COTS hardware. Alternatively COTS hardware have low costs, higher processing power, diverse products but has to be traded off in terms of reliability[4].

ScOSA is the next generation of OBC developed by the DLR German Aerospace Centre to address the growing demand of requirements and complexity in space missions by using a combination of highly reliable computing nodes and high performance COTS hardware, which provides a new possibility of distributed, reliable, fault tolerant and reconfigurable architecture. Several demonstrations like the robotic on orbit servicing and earth observation are used to evaluate the system performance characterstics such as dynamic reconfiguration and FDIR[6].

The current ScOSA system uses a static decision graph to route data between the nodes and trigger a reconfiguration when a fault occurs. The ScOSA system is designed to support upto 20 nodes, maintaining a static graph will become extremely complex and can be prone to implementation errors. The main motivation of this thesis is to research on finding new ways to implement dynamic routing and test them with simulation, considering the constraints of the SpaceWireIPC[7] which handles communication between the nodes[6].

1.2 Thesis Outline

Following the brief introduction to the mission statement and purpose, the thesis is divided in to following chapters. Each of them are divided into separate sections and subsections which

are summarized below.

1.2.1 Chapter 2 - Background

The first section provides a detailed background on the evolution of ScOSA system, its application, purpose and how it can shape up the future of On Board Computers in Spacecrafts. The next section gives an overview of the SpaceWire protocol and its advantages in a distributed network. The third section describes the SpaceWire IPC. In the fourth section the objectives and goals are discussed and finally the last section gives an introduction to current state of art dynamic routing protocols.

1.2.2 Chapter 3 - Design

The first section gives detailed information on the proposed routing algorithm. The next section describes about the OMNeT++ simulation environment and the design section is subdivided into multiple subsections and explains how different components are modelled in the ScOSA node.

1.2.3 Chapter 4 - Implementation and Results

The first section describes how the routing algorithm is evaluated by using a case study and its network activity and a scaled up generic network topology is also implemented to compare the results.

1.2.4 Chapter 5 - Conclusion

This chapter presents the conclusion of the thesis along with some the lessons learned during the development and implementation.

1.2.5 Chapter 6 - Future Work

This chapter provides an outlook to what can other routing strategies can be implemented in the future and different types of metrics that can be used to evaluate them.

Chapter 2

Background

2.1 ScOSA

Space Missions in future will face many technical challenges in several areas with the increase in complexity and the amount of data to be processed. Some examples to be specific, can be due to the improved resolution of sensor systems on satellites which requires more on-board processing like filtering, selection, compression, etc. In missions like robotic explorations, rovers or deep space probes which require higher degree of autonomy the OBC should support complex tasks due to limited bandwidth and communication delay. These complex applications also require real time processing which can limit the design decisions to opt for a conventional radiation hardened OBC, which also needs to be reliable and durable. Moreover the individual subsystems like attitude control or PMU usually has its own hot or cold redundant component meaning upon failure cannot be replaced by a spare unit from another subsystem which creates a scenario where a lot of computational power is unused. OBC-NG was first developed by DLR in 2014 with a TRL 4 to address these issues[8].

The main purpose of the OBC-NG is to provide higher computing performance, higher reliability in a real time environment and reconfiguration of the system using telecommand after different mission phases. There objectives are achieved by designing a re-configurable, distributed on board architecture using different processing components like CPUs, FPGAs, special purpose processors connected via a switched network along with COTS components to reduce the costs and to further increase the performance.

2.1.1 Related Work

Space qualified processors are very expensive and lack performance when compared to conventional components. Unit prices vary between $20,000\$ - 70,000\$$ for a QML-V (which can withstand $50krad$ total dose of radiation hardness) micro controllers and FPGAs compared to less than $100\$$ for a COTS processor[8]. COTS processors are used in space by NASA since

the 80s [9]. This paved the way to use COTS components in the system design with high performance for non critical payloads. A survey of these projects can be found in [10]. One such example of a fault tolerant architecture is BIRD microsatellite developed by DLR in 2001 which uses four PowerPC MPC623 processor as CPUs[11].

The performance of the OBC can also be improved by using several processors in parallel or by increasing the number of cores in a single chip. The introduction of OPERA Maestro processor which has TILE64 architecture of reliable multicore processors was developed to improve the reliability of space applications specific to NASA's constraints[12]. Alternatively cluster based architectures are also very effective but requires high performance communication networks or data buses to connect the processors. The OBC of Iridium NEXT satellite developed by Seakr Engineering uses a three PowerPC-CPUs, six radiation hardened Xilinx- FPGAs and SpaceWire to connect every node[13]. Another approach proposed in [14] uses a distributed network architecture connected by CANbus without a master where processor nodes work independently based on broadcast messages. The DM developed by NASA could achieve $10-100$ times more processing power based on COTS[15] and aims to achieve a TRL7[16].

2.1.2 Core Concepts

ScOSA is developed based on a variety of core concepts and features. These features are explained below[6].

1. **Distributed System:** This architecture provides interconnection of multiple computing nodes which increases the processing power and allows parallel processing of algorithms and distributed tasks allocation.

2. **Network Interconnection:** ScOSA allows flexible network topology and is designed to support at least 20 nodes using packet oriented data exchange allowing point to point connections and fully meshed architecture.

3. **Heterogeneous:** The design does not require all nodes to be identical which implies nodes can have different properties and perform mission specific tasks based on the constraints and tasks.

4. **Reconfiguration:** This allows the operating system to change the state of the system in a collective way which includes network interconnection, node activation and task distribution. A system reconfiguration can be planned by the mission requirements or can be triggered by a failure as a reaction.

5. **Fault Tolerance:** The system has its own mitigation and isolation measures to protect from a mission failure. A hybrid scheme is implemented which allows a combination

of nodes with a variety of fault tolerant models such as fail-safe shown in Figure 2.6. Reliable nodes have fault tolerance to SEU already embedded into them and for COTS several software mitigation techniques are implemented.

6. **Hardware Acceleration:** Some of the nodes are based on FPGAs offering flexibility to run application specific accelerators which increases the throughput and efficiency. The COTS processors offer higher performance than the Reliable nodes using which the accelerator IP core of the COTS can be changed at run time.

7. **Scalability:** The ScOSA system can support upto 20 nodes with several design parameters. A generic network topology will also allow to swap COTS components in future design.

2.1.3 Hardware Design

ScOSA consists of three different types of nodes RCN Reliable Computing Node, HPN High Performance Node and IFN Interface Node. Figure 2.1 shows two RCNs, three HPNs and one IFN connected via SpaceWire links. All IFNs and HPNs conatain a SpaceWire router IP cores which enables data routing between nodes. A laboratory setup of ScOSA can be seen in Figure 2.2. The setup consists of the RCN on the left, three HPNs in the middle and two camera nodes on the right. The cameras and HPNs are connected via Ethernet while the RCN and HPNs are connected via SpaceWire. The HPNs act as a bridge between the two networks.

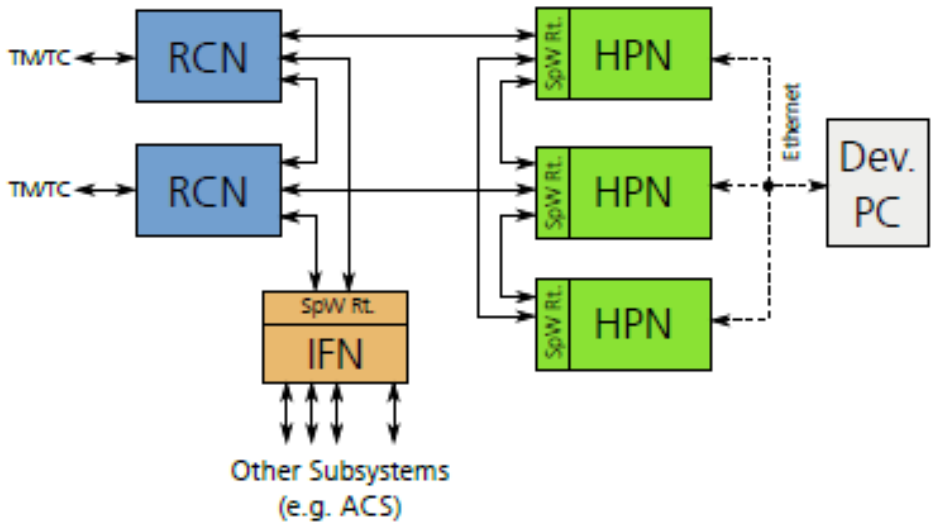


Figure 2.1: Hardware Block Diagram of ScOSA[6]

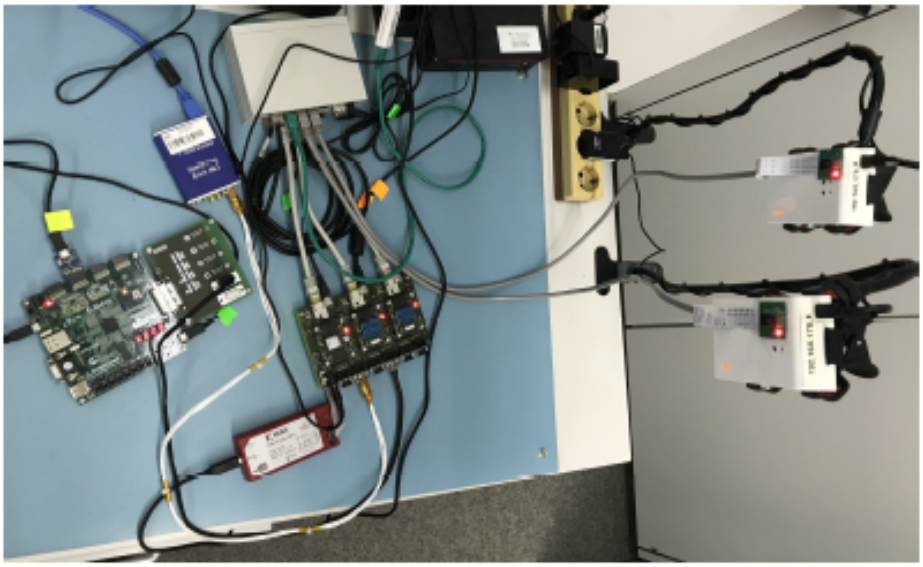


Figure 2.2: Laboratory Setup of ScOSA[17]

A description of hardware selection is summarized below [6].

1. **HPN:** Several COTS components are available, but in ScOSA a Xilinx Zynq is used for its build-in FPGA and fast ARM Cortex-A9 CPU. FDIR algorithms are implemented at the system level and node level in both software and hardware.

2. **RCN:** It is based on a LEON 3 SoC with a flash based FPGA whic supports CCSDS compatible telemetry and telecommand interfaces.

3. **IFN:** Multiple IFNs can be physically distributed in ScOSA system. They are very application specific and in the current setup IFN represents the ACS Attitude Control System in HiL setup.

4. **Network:** Current version of ScOSA uses SpaceWire but in future integration of SpaceFiber is also possible because if offers a higher speed. Gigait Ethernet is available in all HPNs which provides a inter-node link with the development computer. An Upgrade of Ethernet-SpaceWire Protocol is described in [18].

2.1.4 Classification of Nodes

The nodes in ScOSA are of different types and has different roles and can consist of different components. Table 2.1 represents the classification of nodes[8].

Types	Processing Node (PN)	Interface Node (IN)
Hardware Components	Main Processing unit CPU, Router, Optional Co-processor (FPGA, etc.)	Micro-controller, Router, Interfaces to peripherals, Mass Storage
Possible Roles	Master, Observer, Worker	Storage, Interface

Table 2.1: Classification of ScOSA Nodes [8]

The roles for the Master are to control, monitor and distribute tasks but ScOSA does not have a dedicated Master and one of the processing nodes will be assigned as one. Observer will monitor the master and other priority observers and Worker nodes with no management functionalities are used for data processing alone. During runtime roles can also be changed by reconfiguration. More than one role can be assigned to a single node for example the master can also process data along with management functionalities. However, two management roles cannot be assigned to a single node where they will not be able to detect a failure[19].

2.1.5 Software

The ScOSA software stack uses a layered approach to distribute tasks across all available computing nodes, establishes communication via different channels and mitigate potential system faults. These components are shown in Figure 2.3. This hierarchy of data flow within the separated components form a robust and reactive system.

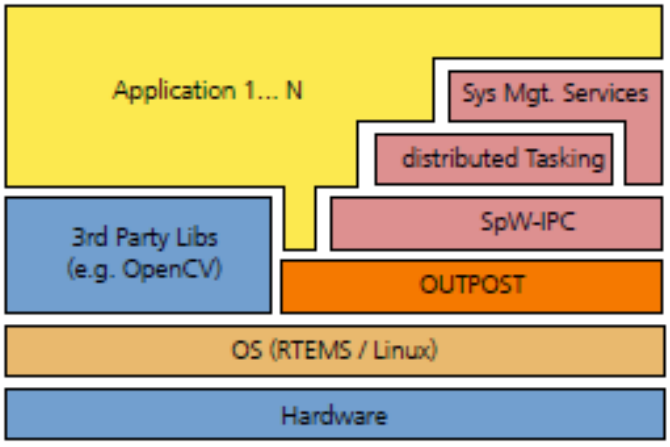


Figure 2.3: Software Block Diagram of ScOSA[6]

The high level functionalities such as reconfiguration, monitoring and other health management tasks are provided at the top layer. The distributed tasking framework offers the developers to integrate their algorithms by task definition and communication which can be changed at runtime. The next layer is the SpaceWireIPC[7] which ensures packet routing. The FDIR algorithms are implemented in this layer and it also supports both SpaceWire and Ethernet

Links. IPC will be explained in detail in the section 2.3. The Software hierarchy also supports third party software and their operating system dependencies. The ScOSA setup supports a variety of hardware components and the OUTPOST library provides the abstraction required.

2.1.6 System Management Services

The main focus of the ScOSA FDIR subsystem is on software implementation and is integrated in the entire software stack on all levels. Figure 2.4 shows the different levels and their causes[20].

Criticality	Software Mechanisms	Cause
Level 4 Safe Mode	• Safe Mode SW Configurations (I)	• Hardware alarms • Multiple level 2 or 3 failures
Level 3 System	• System Monitor (D) • Reconfiguration (I,R) • Reintegration (R)	• Subsystem failure • Inter-node mechanisms
Level 2 Node	• Watchdog Timer (D,I) • Soft Error Signal Handler (D,I)	• Subsystem failure • Local mechanisms
Level 1 Task	• System Alert (D) • Plausibility Check (D) • Checkpointing (R)	• Unit failure • subsystem performance degradation
Level 0 Data	• Voter (D,I)	• Failures with no effect on the performance

Figure 2.4: FDIR levels of ScOSA[20]

In the current version, the Master monitors the other nodes by sending a small message called a Heartbeat(HB) periodically to which the nodes that are alive respond with an Acknowledgement(ACK). Two other processing nodes are also assigned with the role of observers to increase reliability of the system and provide redundancy. Observer one observes the master and Observer two observes both Master and Observer One. A reconfiguration is triggered based on the priority in the order of Master, Observer one and Observer two[19]. A graphical representation is shown in Figure 2.5

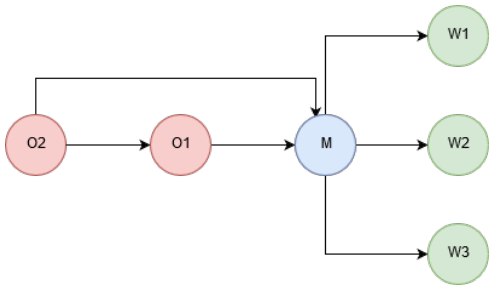


Figure 2.5: Monitoring Service in ScOSA Adopted from [19]

Two types of reconfiguration are possible, planned reconfiguration and reconfiguration due to failure. The main purpose of reconfiguration is either to replace the failed component or to

isolate it from the rest of the system[4]. ScOSA redistributes its tasks to the remaining nodes after a failure is detected. Based on the severity of the fault, nodes stop the data transfer in the network to reduce the reconfiguration time. Reconfiguration is currently based on a static decision graph as shown in the Figure 2.6.

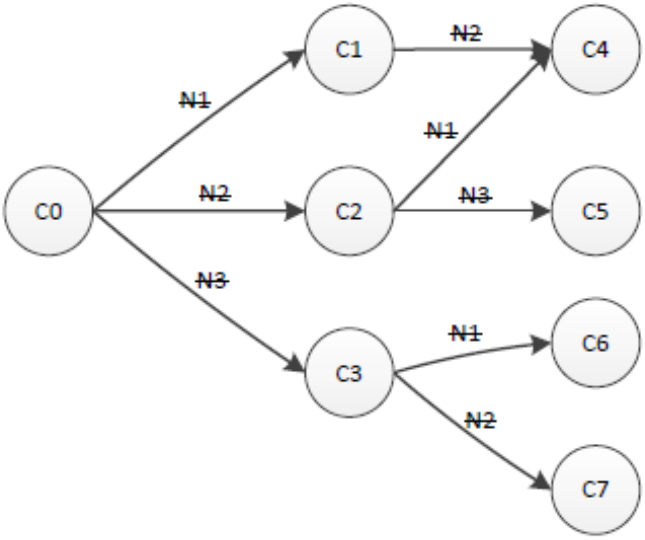


Figure 2.6: Decision graph for reconfiguration in ScOSA[8]

The Master or the highest priority observer parses through the static decision graph to trigger reconfiguration. The graph shows the initial configuration and next possible configuration to be used depending on the failed node. When all node fails the system enters safe mode and all tasks are handled by the Master.

Similarly in the current version of the system, a static routing table is used to route data based on the shortest hop count and the current configuration. Maintaining a configuration file and a routing table for a few nodes might be feasible, but to accommodate more nodes a dynamic routing strategy is required which can be used for both routing data and reconfiguration.

2.1.7 Applications

ScOSA can be used in a variety of space applications some of the examples are Earth Observation, Robotic On Orbit Servicing, ACS and Autonomous Rendezvous Navigation [6].

2.2 SpaceWire

SpaceWire provides reliable data handling capabilities for OBC in a spacecraft. A variety of components such as CPU, mass memories, payloads and other subsystems can be integrated with SpaceWire. The Spacewire protocol is based on IEEE 1355 communication standard[21]

and is developed by ESA in coordination with NASA, JAXA and RKA. Some the important missions to use SpaceWire are ExoMars rover, BepiColombo and James Webb Space Telescope. SpaceWire offers several features, some of the most important ones are[22]:

- Point to Point flexible network architecture.
- High speed of data transfer in the range of ($2\,\mathrm{Mbits/s}$ to $200\,\mathrm{Mbits/s}$) with bidirectional full duplex data links.
- Low power consumption and simple implementation.
- Very low latency
- Low cost and simple implementation and integration with other subsystems.
- Very less buffer memory is used to route data.

SpaceWire offers a flexible network topology using routing switches, point to point links and a router based architecture with full redundancy and fault tolerance. A sample topology is shown in Figure 2.7. This architecture clearly shows the flexibility and interconnection of multiple subsystems. Instrument 1 for example is directly connected to the Mass memory module and the rest of the modules are connected via SpaceWire router. Instrument 4 for example is connected to RTC which has it own CANBus network.

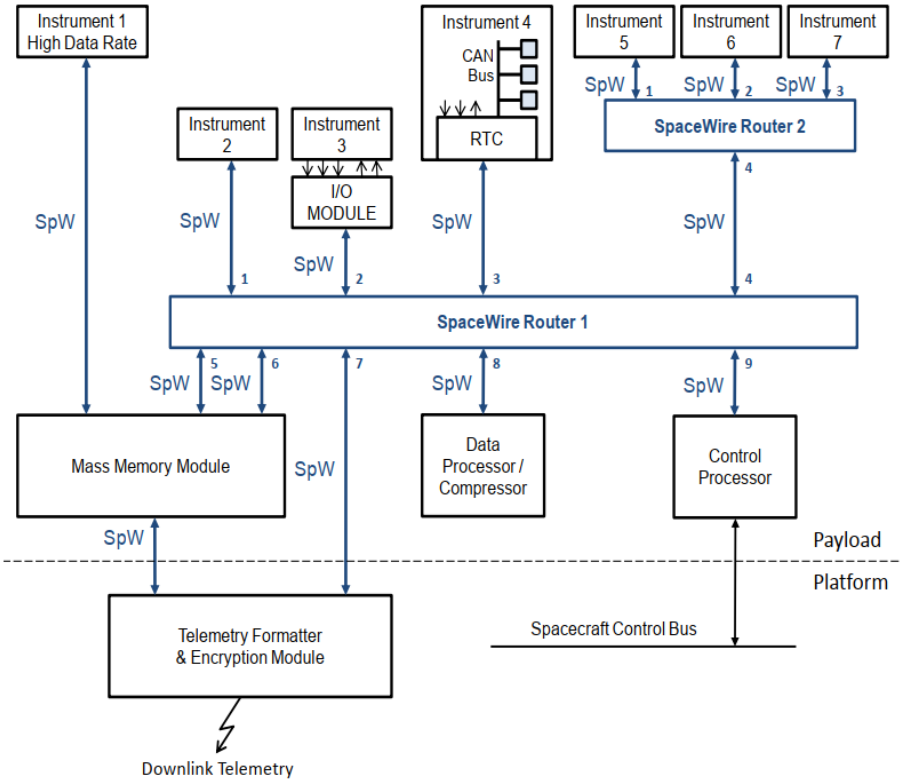


Figure 2.7: Sample Topology of SpaceWire Architecture[23]

The working principle of data transmission is based on Low Voltage Differential Signalling (LVDS) which sends data and strobe signals as serial information on twisted pair of wires. The data handling is taken care by a link state machine which is accessible to the user. It ensures reliable data transfer from nodes by providing a simple mechanism from starting the link to maintaining the link errors by using recovery techniques[23].

2.2.1 SpaceWire Router

SpaceWire packets are sent over the links as information depending upon the availability of the link and the receiver. Each packet contains the destination address, cargo and an EOP End of Packet. The character after an EOP represents the start of the new packet. The packet as a whole is encoded with a number of data characters or otherwise known as flits. There is no restriction on the size of the cargo data. Information is routed based on the destination address. SpaceWire offers two different ways to address a packet. Path addressing is an intelligent way of encoding the complete route the packet has to take in the packet header i.e. in the destination address field otherwise known as source based routing. The other way of routing data is based on the logical addressing which has a static routing table from which routing information is obtained. Figure 2.8 represents the data format of a packet in SpaceWire[23].

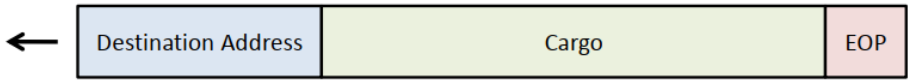


Figure 2.8: SpaceWire Packet Format[23]

The Current ScOSA system uses logical addressing for simplicity reasons but can also support path addressing [7]. All nodes in ScOSA system is built in with a router implemented with a dedicated FPGA. Figure 2.9 shows the high level architecture of a SpaceWire router in ScOSA setup which contains FIFO queues and interfaces to connect with other devices, routing table is used to route data and port handler will request permission from the arbiter before allocating an output port to a packet. An improved SpaceWire router design has been described in [24] along with a comparison study of traditional wormhole switching and the SpaceWire router.

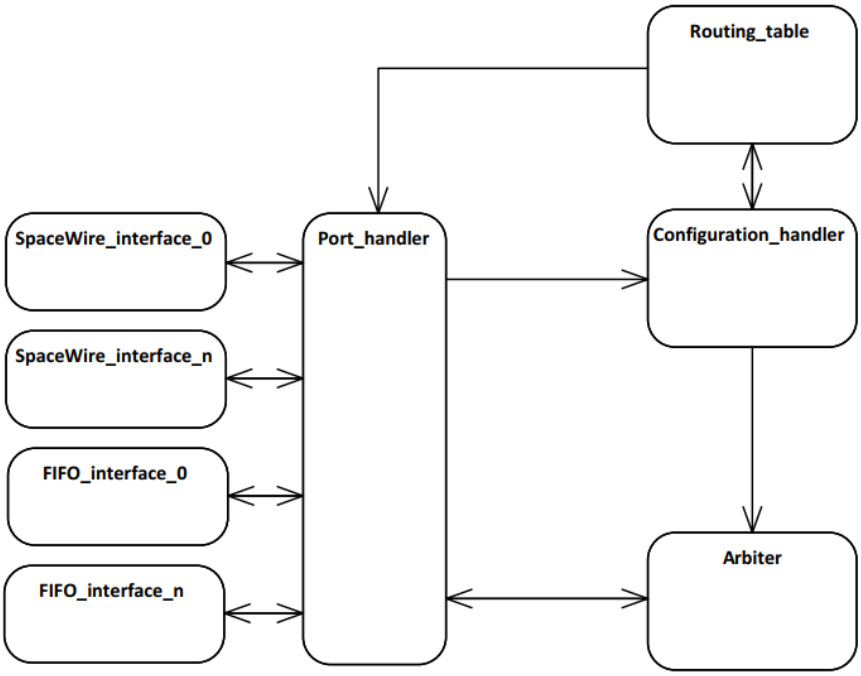


Figure 2.9: High Level Architecture of the ScOSA router[8]

2.2.2 Wormhole Switching

One distinct feature of the SpaceWire protocol is the use of Wormhole routing. Based on the address destination, the router forwards the packet if the next FIFO queue is available. As mentioned in the previous subsection 2.2.1 a packet is divided into number of data flits. The header flit which contains the address will block the FIFO queues for the entire route it travels and the EOP (tail) flit will unblock the queues. This strategy ensures that there is no data buffering before or after switching. Although Wormhole routing has very distinctive advantages it can suffer from deadlock, livelock and starvation conditions[25].

Deadlock conditions occur when two packets are waiting to be forwarded to the same FIFO queue which is not available and ends up blocking the whole route. Livelock condition occurs when a packet keeps travelling through the network without reaching its destination, this reduces the throughput capacity of the link. Finally starvation occurs when a higher priority packet blocks the route where in the worst case scenario the lower priority packet ends up being dropped. SpaceWire protocol allows two types of priority setting, high and low priority which is assigned during packet creation and also depends on the source node. Since there is no restrictions on size of the payload data, a large packet has a higher probability of blocking other packets or being dropped. Care should be taken while designing a routing algorithm to avoid these conditions[23].

2.3 SpaceWire IPC

Several standardized protocols have been developed on top of the existing SpaceWire protocol to transfer data between nodes, the SpaceWireIPC[7] provides inter process communication for a distributed network which can also support reconfiguration on the application level. This feature allows COTS components to be integrated to get high performance. Table 2.2 shows the different SpaceWire protocols available with its features[7].

Features	**RMAP**	**CCSDS**	**GRDDP**	**STUP**	**SpaceWire - R**	**IPC**
Data Correctness	X		X	X	X	X
Data Re-transmission			X		X	X
Multiplexing			X		X	X
Large Message Transmission			X		X	X
Flow Control			X		X	
HeartBeat Monitoring					X	X
Reconfiguration Support						X
Error Notification						X
Publish/Subscribe						X
Request Response	X			X		X

Table 2.2: Comparison of SpaceWire Based Protocols [7]

The IPC plays a vital role in a distributed network setup, for example a bus topology is not scalable and can affect the network throughput. The IPC offers a flexible network topology which in case of ScOSA prototype is an irregular one. IPC is located in the transport layer of the ISO-OSI model. IPC supports some core features like management services, multiple logical nodes on one physical device, reliable transmissions of data, FDIR fail safe mechanisms, large message transmission and transparency in using underlying protocols. IPC is a message based protocol where messages are encoded in packets of data which can be sent between nodes. The sequencing technique used allows splitting of larger packets into user defined smaller ones and reassembling them at the destination. IPC also offers reliable data or unreliable data, each reliable message is bidirectional with ACK and a checksum to verify the integrity. Figure 2.10 shows the structure of the spaceWire packet[7].

Target SpW Address (1 byte)	.....	Target SpW Address (1 byte)
Target Logical Address (1 byte)	Potocol Identifier (1 byte)	Sender Node ID (2 bytes)
Receiver Node ID (2 bytes)	Timestamp (8 bytes)	Message Type (1 byte)
Payload Data (0 to n bytes)	Checksum (4 bytes)	EOP (1 byte)

Figure 2.10: Structure of a SpaceWire Packet in IPC [7]

The structure of the message is same for all message types it contains the source address, destination address, timestamp, type of message, size of the payload data and checksum. Summary of message types supported is shown in Table 2.3. A detailed description of these message types are explained in [7].

Integer Value	**Message Type**
0	Unreliable Data Transmission
1	Reliable Data Transmission
2	Data Request
3	Data Response
4	Reconfiguration Request
5	Message Acknowledgement
6	HeartBeat
7	Error Notification
128+	Large Message Transfer

Table 2.3: Summary of Message Types supported in IPC [7]

The HeartBeat(HB) is an important periodic message which offers reliable information, the dynamic routing algorithm proposed in this report is based on the HB message. Different types of HB messages are possible based on the user configuration. Traditional approaches such as the PULL model and PUSH model described in [26] of the HB messages are supported. The PULL type follows the request and acknowledgement model while the PUSH type follows the publish/subscribe model. Monitoring mechanisms can be classified based on their error detection level[27], in the current ScOSA setup it is based on the node level or the stay alive status of the nodes. The PULL model is used in ScOSA to provide a deterministic behaviour.

2.4 Goals and Objectives

The current ScOSA prototype uses the logical addressing to route data between nodes which implies that all messages in the network uses the static routing table to reach their destination. The routing table in the SpaceWire router is configured and reconfigured using the RMAP protocol based on the reconfiguration request[7] and decisions are taken based the graph as shown in Figure 2.6.

However the current static routing table has its own limitations such as maintainability and creating such configurations can be very tedious and error prone especially when the number of nodes increases. Another problem is the link utilization in static routing where all links are not used optimally, some links are overloaded with data during critical mission phases while other links tend to be in idle condition. The motivating factor is to design and develop a dynamic routing algorithm which not only routes data in a optimal way but also improves other metrics like reducing end to end delay of a packet. Another challenge will be to route data in a safe and deterministic way irrespective of the network topology. The core requirements for the dynamic routing protocol are listed below:

- The routing protocol should autonomously route data in runtime without an user interference.
- Upto 20 nodes are supported in ScOSA, the routing algorithm should be scalable irrespective of the network topology.
- Reliable data should be routed in a deterministic way.
- Quality of Service like network latency, end to end delay should be implemented in the routing algorithm.

In this thesis a new dynamic routing strategy is proposed, implemented in simulation and analysed based on literature research and existing spaceWire protocols.

2.5 State of the Art

Dynamic routing implies that data is being forwarded from a source to destination through an optimal route considering the current environment conditions of the distributed system. The routing algorithm can take its decision based on several metrics like shortest path with minimum hops or link state where a cost function is calculated before a routing decision is taken. Several routing strategies have been proposed over the years, the most famous ones are the Routing Information Protocol (RIP) based on Bellman-Ford algorithm (1969) which is a distance vector routing strategy and the other one being Open Shortest Path First (OSPF) based

on Dijkstra's algorithm(1980) which is a link state routing strategy. A performance comparison of these routing algorithms are shown in [28]. Another important routing challenge was in the Wireless Ad Hoc networks (WANET) for which proactive and reacting routing strategies were developed. A comparison of these routing strategies can be found in [29]. These routing algorithms are designed to address an unknown network topology where it is connected to an unknown number of nodes and usually packet loss in such systems does not have any significant impact. Other parameters such as latency and deterministic routing is also not critical.

Packet Switching networks also face problems due to congestion of data packets and there are several methods to address them such as buffer flow control, schedule based control, rate based control, etc. They address how data flow can be improved in a distributed system but these techniques cannot be used to route data. Several congestion control strategies are discussed in [30].

Network on Chips NoC use wormhole switching and have several dynamic routing techniques and flow control mechanisms using virtual channels available[25]. However they have their own constraints such as using a fixed topology and can be used only for in chip communication. Moreover SpaceWire does not support the use of virtual channels due to high cost of adding additional buffers in the SpaceWire router.

Although there are several dynamic routing strategies available, they are clearly not suitable for a distributed OBC like ScOSA which has a unique network topology involving safety critical functionalities and operates in a different environment.

Chapter 3

Design

The core requirements of the dynamic routing algorithm is defined in section 2.4, upon further research on literature, similar applications and distributed systems discussed in section 2.5, a new routing algorithm is proposed based the SpaceWire IPC which can meet the requirements without affecting the existing protocol.

3.1 Dynamic Routing Algorithm

A brief introduction on how inter process communication can be achieved using SpaceWire by offering several features such as management services and monitoring is discussed in section 2.3. In traditional distributed networks, the master node sends a short periodic control signal called the HeartBeat (HB) to which all the other nodes respond with an Acknowledgement (ACK) also known as PULL type of HeartBeat. This mechanism helps monitoring the overall health status of the distributed network and when a node fails to send ACK signal the master node can initiate reconfiguration or trigger FDIR algorithms based on the severity. The current ScOSA system already has a PULL type HB implemented.

The new proposed algorithm will replace the traditional HB signal with a weight matrix which contains information on link utilization. This matrix will be sent from the master node and all other nodes will reply with how much data it has received during the last HB interval and from which source instead of just sending an ACK. This is an iterative process which means for every HB interval the master node will internally calculate the weights based on the replies it gets from other nodes and updates the weight matrix before sending it in the next cycle. This updated matrix is used by all other nodes to make routing decisions. This routing algorithm is heavily influenced by the traditional link state routing which is a source based proactive routing strategy meaning all nodes in the network either have full information about the network activity or at least with its nearest neighbour before taking a routing decision. One major disadvantage of link state routing is the algorithm will flood the network with broadcast information and calculates

the optimal route based on the response[31]. Flooding the network in space environment could lead to total failure of the spacecraft and it is therefore not recommended. Also for any given space mission the network topology is already known and the flight software will be tested and simulated under all scenarios before launch.

The SpaceWireIPC supports source based addressing which means the nodes set the whole route a packet has to take before sending them. Using the proposed algorithm, all nodes will have the entire link utilization of the network but there is one disadvantage because only the master node has the current weight matrix and all other nodes have the weight matrix from the previous update. Different components involved in the design of the algorithm are discussed in the following subsections.

3.1.1 Weight Matrix

The weight matrix has a direct correlation with the network topology. It is formed based on number of nodes and links present in the network. Although ScOSA has no defined network topology, information on nodes and its interconnection will be known when a mission is planned. Figure 3.1 shows the weight matrix for ATON configuration, a case study considered in evaluating the routing algorithm and is discussed in the section 4.1

```
int topology[4][4] = { {0, 1, 1, 1},
                       {1, 0, 0, 1},
                       {1, 0, 0, 1},
                       {1, 1, 1, 0},
                     };
```

Figure 3.1: Weight Matrix of ATON Configuration [32]

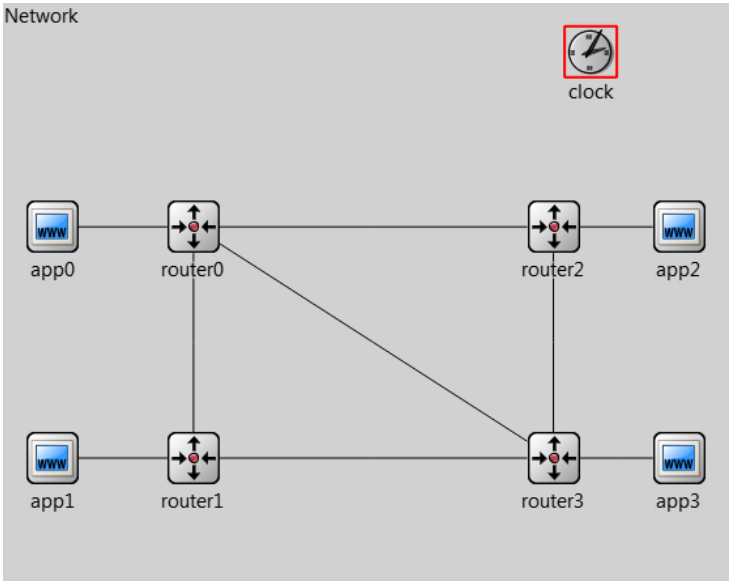


Figure 3.2: ATON Network Representation

The graphical representation of the ATON system can be seen in Figure 3.2. An application and a router forms a node. The diagonal elements $a_{00}, a_{11}, a_{22}, a_{33}$ represent the nodes and the other elements in the row represents the links it is directly connected, when there is a 0 in the matrix other than diagonal elements it means there is no direct connection with that Node. For example when Node 0 sends $10\,\text{bytes}$ of data to Node 3 element a_{03} will be updated with $10\,\text{bytes}$ similarly when Node 3 sends $12\,\text{bytes}$ of data to Node 0 element a_{31} will be updated with $12\,\text{bytes}$ of data making the weight matrix bidirectional. This a flexible way of representing any given network topology in the form of a matrix.

3.1.2 Link Utilization

Link utilization is the most important metric in the algorithm which makes dynamic routing possible. It is based on a very simple formula.

$$Link\,Utilization = \frac{Total\ Data\ Received}{HeartBeat\ Interval} \tag{3.1}$$

For example when Node 1 sends $10\,\text{bytes}$ to Node 2 via Node 0 and Node 3 sends $20\,\text{Bytes}$ of data to Node 0 via Node 1 in one cycle of HB. Element a_{10} will be updated with $30\,\text{Bytes}$, element a_{03} will be updated with $10\,\text{bytes}$ and element a_{32} will be updated with $20\,\text{bytes}$. In this way the weight matrix is updated using the link utilization by simple incrementation.

3.1.3 Routes Generation

Since the network topology is already known for a space missions, possible routes can be predefined. Although when the number of nodes and links increases it can be very difficult to maintain. This will be a design decision which has to be considered while route generation and it is safe to limit number of possible routes. For example in the ATON topology shown in Figure 3.2 all nodes have 3 three possible routes to reach their destination. A routing table based on the weight matrix is designed to generate all possible routes. Each node will have a copy of the routing table and the weight matrix.

3.1.4 Optimal Route Selection

The updated weight matrix is used to calculate the score for each route when the data has to be transferred from a source to destination. For any given source and destination the optimal route is calculated based on the simple logic shown in Figure 3.3 by looping through all possible routes.

```
if (currentScore < previousScore){
    use currentScore;
    else
    use previousScore;
}
```

Figure 3.3: Logic to select Optimal Route

3.2 Introduction to OMNeT++

OMNeT++ is an open source discrete event simulation framework based on C++ with a generic architecture. It provides a platform to model and simulate different types of domains such as communication networks, protocols, hardware architecture and performance evaluation of complex software systems. OMNeT++ models consists of modules which can communicate by passing messages. Each active component in the module is a simple module and they can be grouped together to form a compound module. These modules are connected by a channel and a group of modules form a network. Figure 3.4 shows simple modules, compound modules and the arrows represent gates and channels. Message triggered and event triggered mechanisms are supported in the framework.

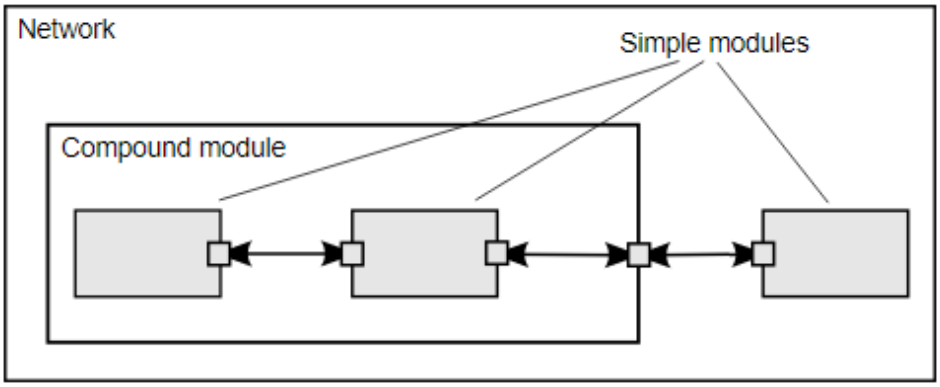


Figure 3.4: Block Diagram of OMNeT++ Sample Module[33]

The main functions of the class cSimpleModule are initialize, invoked by the simulation kernel once at the beginning and handleMessage when a message arrives. Messages, packets and events are all represented by cMessage objects, to schedule future events in order to implement timers, timeouts, and delays a self message can be triggered. Since the simulation kernel is event based, the event can be scheduled at different points in real time to the same simulation time[33]. There are several component specifications in the OMNeT++ framework. The core components are listed below:

- **.ned** file: Network Description is a high-level language that describes the structure of the

modules at every level, from simple module to a model. It also describes the parameters, gates, and network topology of the model.

- **.msg** file: Various types of messages and packets are defined here and OMNeT++ will translate message definitions into C++ classes.
- **.ini** file: Contains configuration of the model and parameters with respect to the simulation runtime.
- **.cc** file: Contains the logic behind each module and can be programmed in standard C++.

The simulations are run using Tkenv user interface which is a GUI toolkit and graphical simulation environment. Cmdenv is a command-line user interface for batch execution[33]. The output of simulations can be recorded as vector or scalar. Vector output is recorded during the simulation into output vector file (.vec). The scalar values are collected during the simulation in a variable and recorded at the end of the simulation in output scalar file (.sca). If the simulation is run with record event log option, the sequence chart is produced and stored in event log file (.elog). It contains the graphical view of modules, events and messages sequence and transmission duration. The first three types of output files can be exported as Scalable Vector Graphics file (.svg). OMNeT++ also allows results to be exported into other working environments such as Python, MATLAB for detailed analysis.

3.3 ScOSA Node Design

Prior to the development of SpaceWire several OEMs had their own proprietary Ad Hoc inter unit communication between the different components which resulted in extended project timeline, difficulty with integration and increased costs. The main objective of SpaceWire was to address these issues by defining a standard protocol which can be flexible and support plug and play devices. For example a data handling system developed for an optical instrument can be replaced by a radar and the standard protocol should be able to support it making it reusable. These standards are clearly defined in the ECSS-E-ST-50-12C Space Engineering handbook. The ScOSA nodes are modelled based on the SpaceWire requirements, they contain the important features such as the CPU, encoder, decoder, routing switches, links, networks and most importantly wormhole switching queues [22]. All the components are modelled in OMNeT++.

All ScOSA nodes contain an application and a router, both of them are compound modules which implies they contain various simple modules. All components modelled have their own functionalities. Some of the design decisions taken during the implementation are listed below, these decisions do not affect the functionality of the routing algorithm.

- All channels have the same data rate and the same delay.

- Only one flit or packet can be processed for a clock cycle.
- Source and Destination nodes process data without any delay.
- All nodes have low priority address so there is no arbitration in the router, it is served in the order of arrival.
- Delays caused by Time Codes and Flow Control Tokens are not modelled.
- Parity bit checking is not implemented for simplicity reasons.
- Header deletion is not implemented.

3.3.1 ScOSA Application Design

All Applications contain a CPU, encoder, decoder and an output queue. Figure 3.5 shows the model of an Application in OMNeT++. These components are essentially defined as classes in object oriented programming and a simple module in OMNeT++. Ideally the Encoder and Decoder should serialize and deserialize packet to flits and vice versa and the CPU should take the routing decisions when source based addressing is used[22]. For simplicity reasons and easy implementation, the designs have been modified and explained in the following subsections.

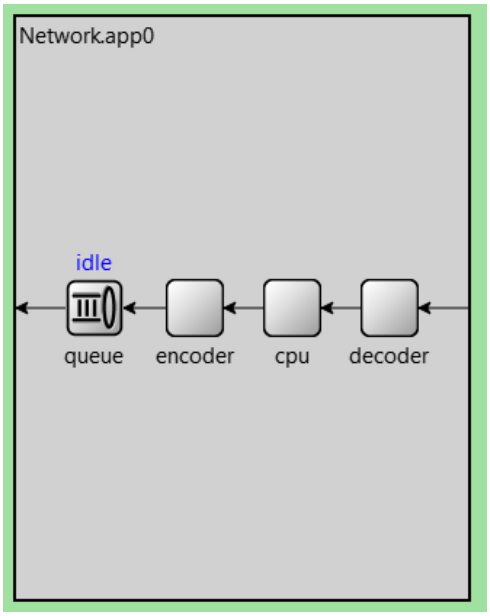


Figure 3.5: ScOSA App Design

CPU

In the current design the CPU's core functionality is to send periodic HBs and data as per the requirements. In OMNeT++ data is sent from one module to another in the form of messages, in this design it's in the form of packets which is a subclass of message. Figure 3.6 shows the

functions handled by the CPU. Initialize and handleMessage performs the core functionalities of each simple module. The handleMessage is split into four different messages based on the packet type and how they are handled when they arrive. Further to that the CPU also serializes and deserializes the packets it is sending and receiving. The resetBuffer() function clears the weight matrix after every HB as precautionary measure to prevent from overflow. The CPU also calls the updateWeightMatrix() and updateSelfMatrix() functions from the encoder during every HB cycle. The new packets created from the CPU are sent to the Encoder and received from the decoder. The CPU is also synchronized with the system clock.

```
void initialize() override;
void handleMessage(cMessage *msg) override;
void sendHeartBeat();
void sendData();
void handleHbRequest(Packet* packet);
void handleHbReply(Packet* packet);
void handleData(Packet* packet);
void handleHbInit(Packet* packet);
void serializeMatrix(Packet* packet);
void deserializeMatrix(Packet* packet);
void printSerializeMatrix(Packet* packet);
void printDeserializeMatrix(Packet* packet);
void resetBuffer();
```

Figure 3.6: Functionalities of the CPU

Encoder

The Encoder also has a lot of important functionalities, most important among them is to convert a packet into several flits based on their packet size. Each flit is modelled to a size of $1\,\mathrm{byte}$. It creates the head flit, data flits and the tail flit. The intelligence of the routing algorithm is modelled in the Encoder. It contains the weight matrix, generates static routes to form a routing table and selects the optimal route. When the Encoder is initialized it statically generates all possible routes with 3 hops for a given source node to a destination node along with its address and output gates. When a packet arrives at the encoder it is converted to flits and based on the weight matrix the optimal route is chosen and sent to the output queue. Figure 3.7 shows the functionalities of the encoder implementation in OMNeT++.

```
class Encoder : public cSimpleModule
{
  public:
    Network * network;
    Encoder();
    std::vector<Route *> getRoutes(int source, int destination);
    Route* getOptimalRoute(int source, int destination);
    void updateWeightMatrix(int source, int destination, int dataSent);
    void resetMatrix();
    void updateSelfMatrix();
  protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    void printRoutes();
    void printOptimalRoute(Route* route);
    std::vector<std::vector<std::vector<Route * > > > routingTable;
};
```

Figure 3.7: Functionalities of the Encoder

Decoder

The Decoder has a simple functionality, it converts all flits into a single packet and sends it to the CPU.

3.3.2 ScOSA Router Design

The high level architecture of the SpaceWire Router is discussed in subsection 2.2.1. The Routing table is implemented in the encoder and the arbiter functionality is not implemented which means all packets have the same priority. However, this decision will not affect the routing algorithm proposed because the amount of data on the network will still be the same with arbitration and does not have any impact on the link utilization. In this design the router is a compound module consisting of a routing switch and output queues based on the number of ports. Figure 3.8 shows how router1 in Figure 3.2 is modelled. The switch is connected to 3 ports where queue[0] is connected to app1, queue[1] is connected to router0 and queue[2] is connected to router3.

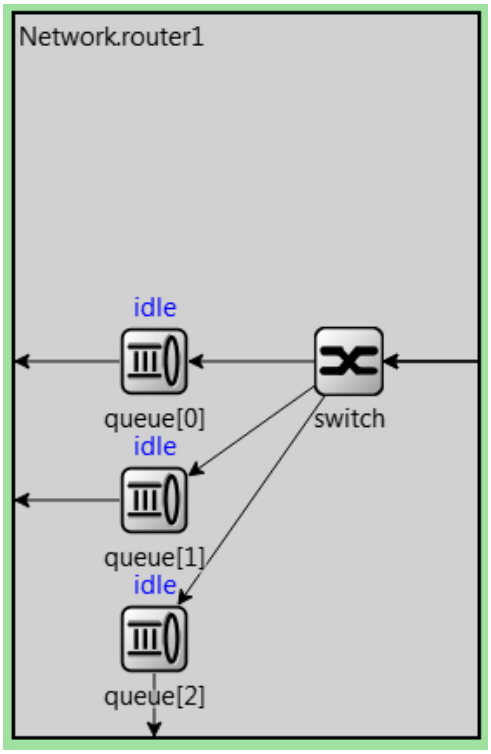


Figure 3.8: ScOSA Router Design

Routing Switch

In the current design the crossbar switch in the SpaceWire router is replaced by a simple switch. Its only functionality is to send data to the output gate mentioned in the head flit which contains all the routing information.

Queues

Another important component in the router design is the Queue, all queues are synchronised with the clock and can process one flit per clock cycle. List of functionalities of the Queue is shown in Figure 3.9. The recievesignal() function subscribes the queue with the clock. The wormhole switching logic is implemented in the popQueueAndSend() function, the head flit checks whether the queue is occupied or not if it is not occupied it blocks the Queue until the tail flit unblocks it. The next queue is also accessed by the head flit and based on the availability and current status of the next queue flits are forwarded.

```
void initialize() override;
void handleMessage(cMessage *msg) override;
void popQueueAndSend();
void receiveSignal(cComponent* source) override;
```

Figure 3.9: Functionalities of Queue

3.3.3 Packets

Packets are essentially messages in OMNeT++. They auto generate a list of basic functionalities such as timestamp, bit length and byte length with the get and set functions. Each packet in this design contains a list of parameters shown in Figure 3.10. Packets are classified into different types, HB request, HB reply and data. All packets are handled at the node level.

```
packet Packet
{
    int srcAddr @packetData;
    int destAddr @packetData;
    int type @packetData;
    int data[1024] @packetData;
    int sendData[7] @packetData;
    int receiveData[7]@packetData;
    simtime_t creationTime;@packetData;
}
```

Figure 3.10: Data in Each Packet

3.3.4 Flits

Flits are generated by the Encoder, all information available in the packets are mapped to flits and they also have additional information such as routes, number of hops and unique ids. All flits are handled at the router level.

```
packet Flit
{
    int address[10];
    int addressLength;
    int data[1024];
    int hops;
    int type;
    int type_packet;
    int sourceId;
    int destinationId;
    int size;
    int uniqueId;
    int sendData[7];
    int receiveData[7];
    simtime_t flitCreationTime;
}
```

Figure 3.11: Data in Each Flit

3.3.5 Channels

Channels are modelled in the Network Description(NED) language which is based on C++ notation. Figure 3.12 shows the data rate of each channel or link and unit delay. Each flit takes $0.1\,\mu s$ to pass through a channel.

```
channel C extends DatarateChannel
{
    delay = 0.1us;//set clock 50 ns
    datarate = 50Mbps;
}
```

Figure 3.12: Channel Settings

3.3.6 Clock

The clock emits reliable time throughout the simulation. Modules can subscribe to the clock. The clock frequency can be set at the beginning of the simulation.

Chapter 4

Implementation and Results Analysis

4.1 ATON Case Study

Since 2010 DLR is working on the Autonomous Terrain-based Optical Navigation project which provides a platform to develop autonomous navigation of spacecraft in orbit, around and during landing on celestial bodies like the Moon, planets, asteroids and comets[34]. This project currently uses an OBC with a single core which handles all the tasking functionalities. Another research project from DLR was to map these tasks to distributed computing nodes, in this case ScOSA[32]. The tasks are mapped based on the satisfiability modulo theories (SMT) solver. The tasks are distributed in four nodes as shown in Figure 3.2. The network has 5 bidirectional channels for inter communication. An application and a router forms a node.
There is no routing required as the topology is small and all nodes are directly connected with other nodes it communicates. The network activity is shown in the Table 4.1.

Source	**Destination**	**Time Period in** ms	**Data Size in** $bytes$
0	2	200	1048614
3	1	200	1048614
2	0	800	146
1	0	800	146
0	1	10	350
0	2	10	350
0	3	10	350
3	0	10	18

Table 4.1: Network Activity in ATON[32]

There is no communication between Node 2 and Node 3 but an exclusive channel exists which is always idle. All nodes transfer data periodically, normally in a distributed systems the period of the HB signal is around $100\,\mathrm{ms}$ to $200\,\mathrm{ms}$. The proposed algorithm cannot be effectively

used because when a the HB interval is set for $200\,\mathrm{ms}$ it means the weight matrix will be reset for 4 cycles and that data sent every $800\,\mathrm{ms}$ will not be captured. Moreover for a static one to one node mapping there will be no deadlocks in the wormhole and the algorithm cannot be fully evaluated. Considering these factors the a new network activity is implemented to mimic the original network activity and time periods have been changed to introduce blocking in the queues as shown in Table 4.2. In order to take the $800\,\mathrm{ms}$ tasks into consideration for dynamic routing the weight matrix is not reset for during the whole simulation. The data size of the HB message is very small so it is not considered for updating the weight matrix, ideally $2\,\mathrm{bytes}$ of data will be required to represent each link. The weight matrix of this configuration is shown in Figure 3.1 at initialize() all links have a weight of 1,

Source	**Destination**	**Time Period in** μs	**Data Size in** $bytes$
0	2	49.95	50
3	1	49.95	50
2	0	74.95	8
1	0	74.95	8
0	1	4.95	10
0	2	4.95	10
0	3	4.95	10
3	0	4.95	5

Table 4.2: Implemented Network Activity for ATON

The use of FIFO queues allows the possibility of using two different clocks, one for the application and another for the router which can operate at a much higher frequency but to maintain easy implementation only one clock is used. Simulation parameters are shown in Table 4.3.

Parameters	**Unit**
Clock Frequency	$20\,\mathrm{MHz}$
Channel Delay	$0.1\,\mu\mathrm{s}$
Data Rate	$50\,\mathrm{Mbps}$
HeartBeat Interval	$25\,\mu\mathrm{s}$
Samples Taken	46
Simulation Time	$0.001\,15\,\mathrm{s}$

Table 4.3: Simulation Parameters for ATON

4.1.1 ATON Results Analysis

The results obtained are a comparison of both the original mapping used in [32] and the proposed dynamic routing for a period of $0.001\,15\,\mathrm{s}$ and $46\,\mathrm{samples}$.

Figure 4.1 shows the total raw data over each link using static routing and it can be clearly seen that only 5 links are effectively used. The graph shows a gradual increase in data because the buffers are not reset to consider all data.

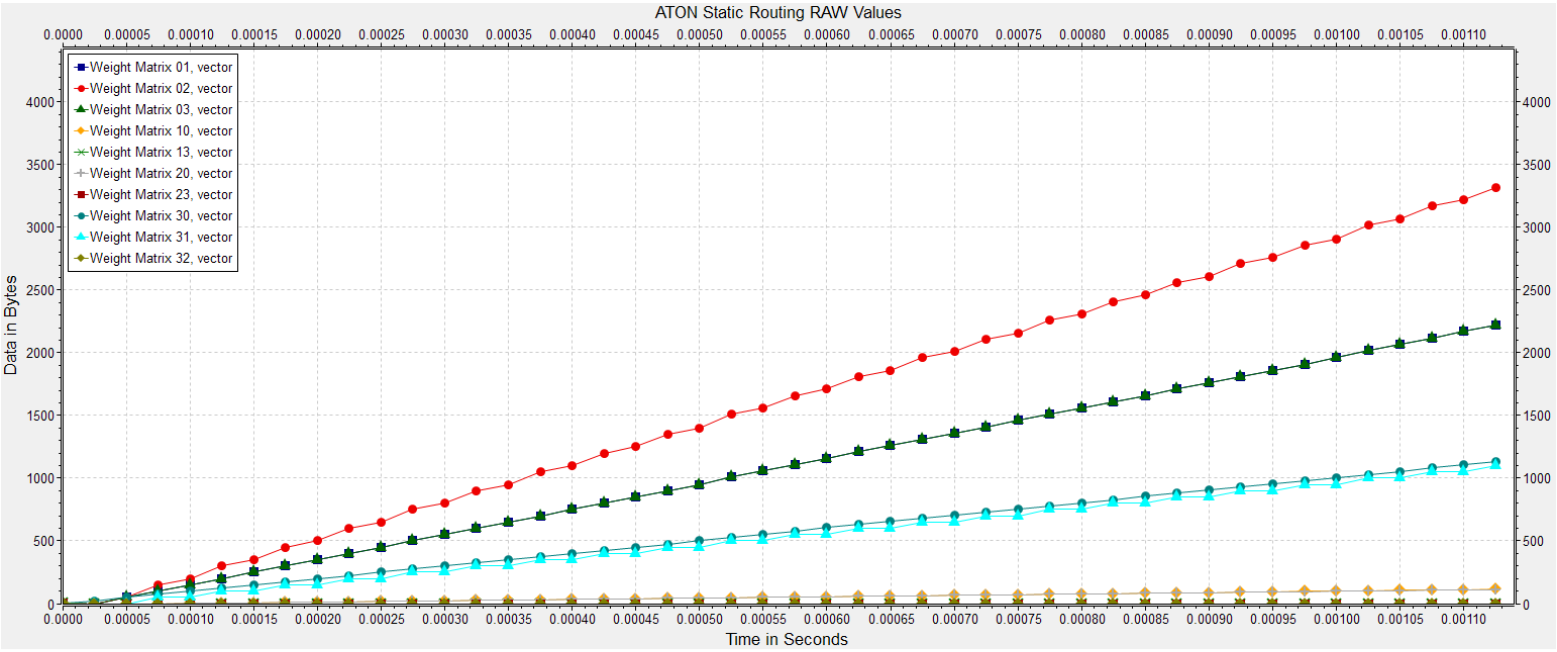


Figure 4.1: ATON Static Routing Raw Values

Figure 4.2 shows total raw data over each link using dynamic routing and oscillations seen in the graph shows that a new route is taken after each HB interval. Another important observation is that total data is almost double in this graph which means effective routing when compared to static routing. Although there is more data in the links, it has no correlation with latency. Wormhole switching is specifically designed to reduce network latency. Note that the Y Axis scales are different in both the graphs.

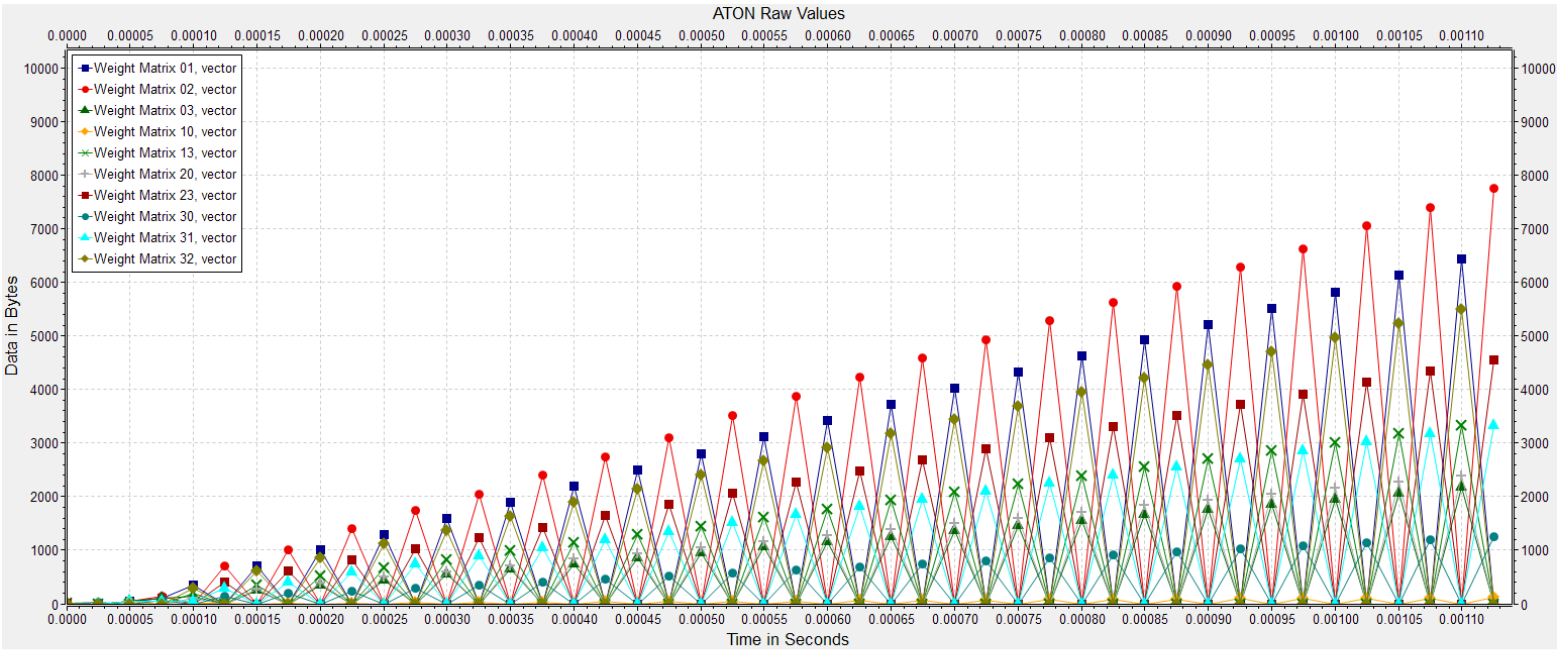


Figure 4.2: ATON Dynamic Routing Raw Values

Figure 4.3 shows the mean values of data on each link over the simulation period with static routing and Figure 4.4 shows the mean values of data on each link over the simulation period with dynamic routing. This also shows that the mean data is quite close to that of static routing in each link but all links are effectively used. Note that the Y Axis scales are different in both the graphs.

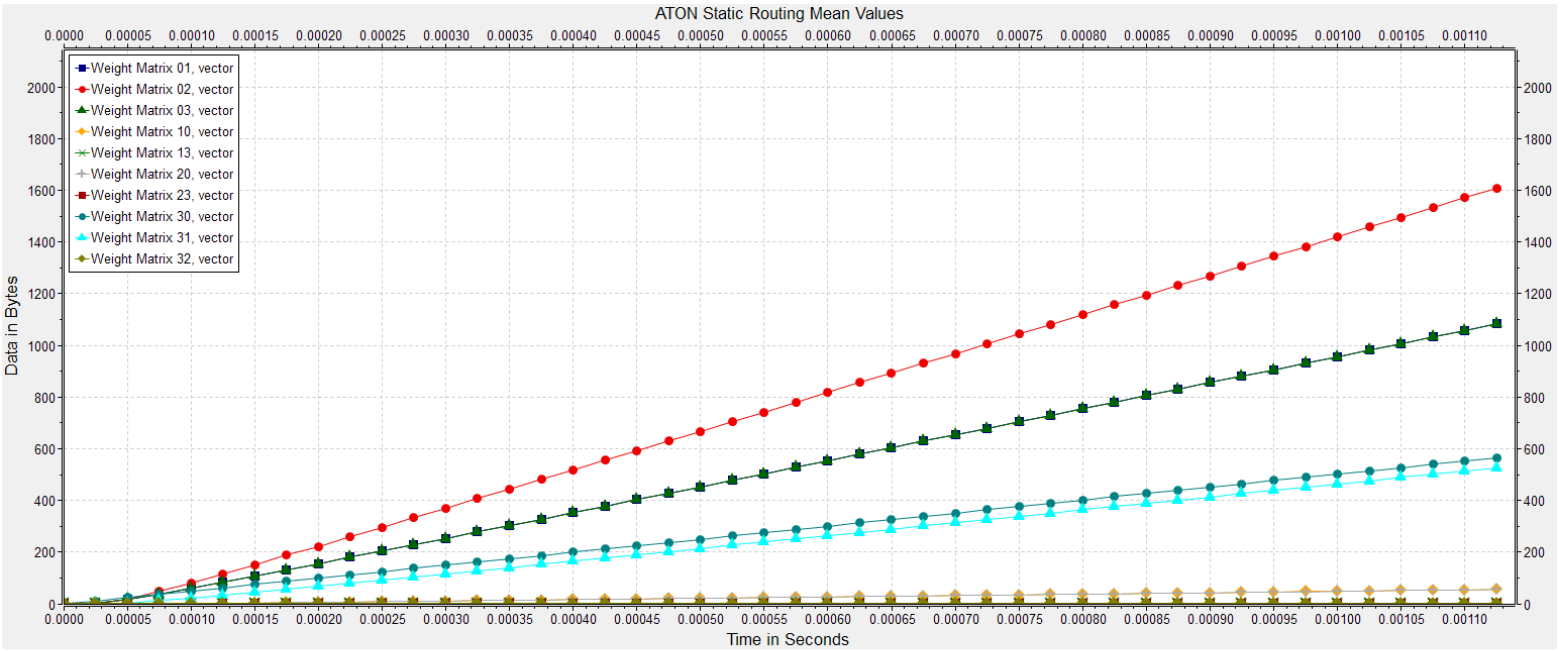


Figure 4.3: ATON Static Routing Mean Values

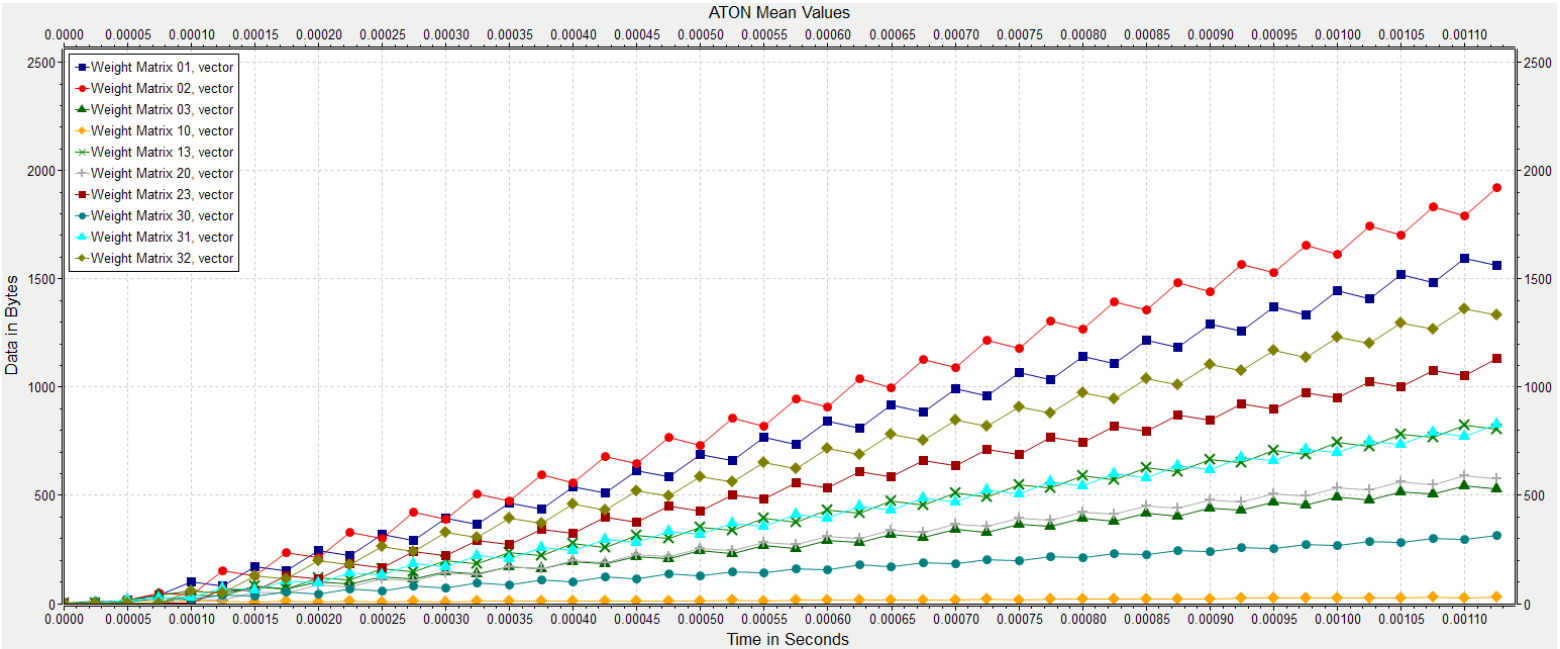


Figure 4.4: ATON Dynamic Routing Mean Values

Figure 4.5 shows total sum of data in each link using static routing and Figure 4.6 shows total sum of data in each link using dynamic routing. It can be clearly seen that all links are used effectively. Note that the Y Axis scales are different in both the graphs.

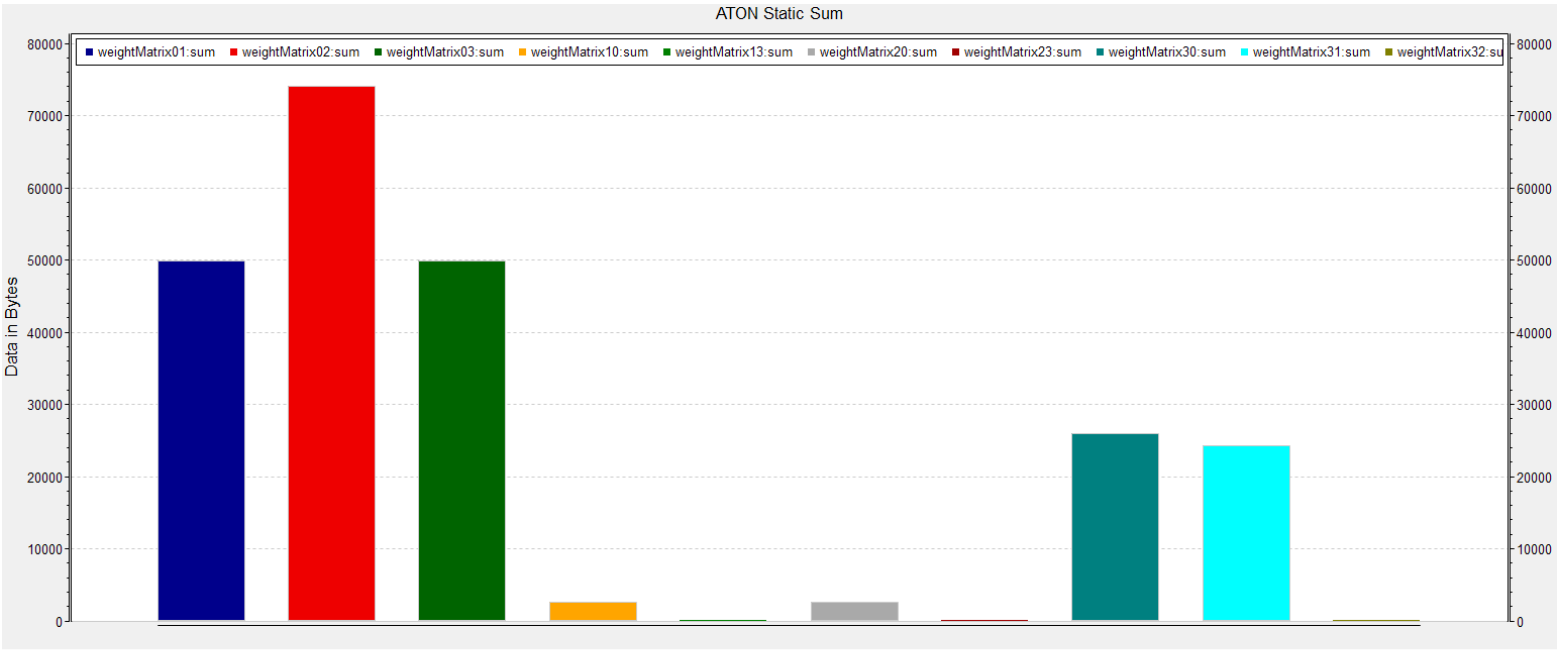


Figure 4.5: ATON Static Routing Sum of Data in Links

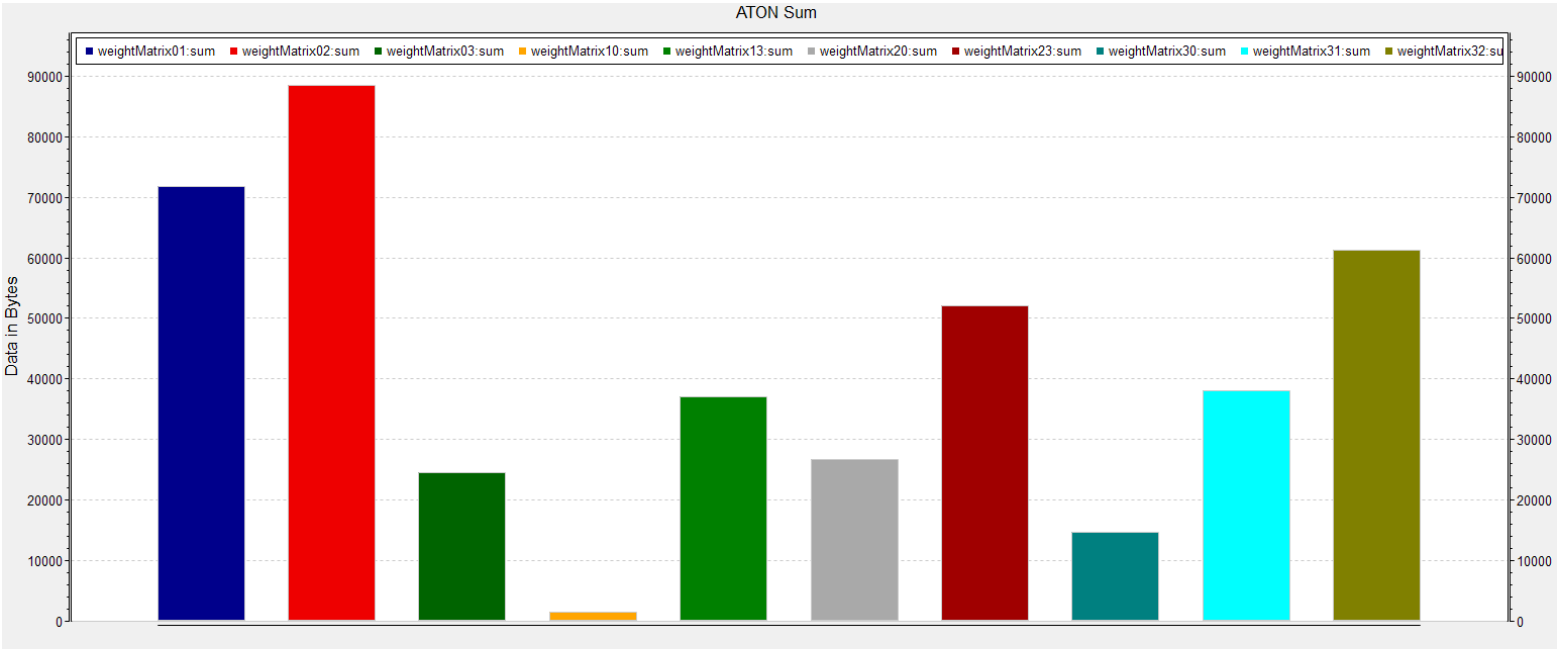


Figure 4.6: ATON Dynamic Routing Sum of Data in Links

In this case study the results show that the dynamic routing algorithm is effective but after every HB cycle there is a new optimal route taken which keeps repeating in a pattern and it can be concluded that even though the routing algorithm works it is not necessarily better than the

static routing.

4.2 Generic Random Topology

The second case study is to scale the network to increase the number of nodes and links. Figure 4.7 shows the generic random topology with 7 nodes and 10 links. The network activity is defined in a stochastic way to evaluate the effectiveness of the routing algorithm and also by comparing the end to end delay of packets from a source to destination. A detailed report on how worst case end to end delay can be calculated in a SpaceWire network is shown in [35]. A case study on time triggered data over SpaceWire for distributed systems is presented in [36]. The latency is calculated using Equation 4.1

$$latency = \frac{bits * links}{txfreq} \tag{4.1}$$

where,

- latency: Duration of the transmission until completion [µs].
- bits: Number of bits required to be transmitted.
- links: Number of links to be traversed.
- txfreq: Average transmission rate of all links involved $[Mhz]$.

and the total latency is calculated using the equation Equation 4.2

$$latency_{total} = latency_{tx} + latency_{unit} + latency_{wait} \tag{4.2}$$

In this implementation the end to end delay is calculated by recording the packet creation time at the source and the packet arrival time at the destination and subtracting them. Although in this model it can be mathematically represented by Equation 4.3

$$\begin{aligned} end\,to\,end\,delay \;=\; &(channel\,delay \;*\; no\,of\,links) \;+\; (bytelength \;*\; clock\,freq) \\ &+ (clock\,freq \;*\; no\,of\,switches) \;+\; (time\,in\,queue) \end{aligned} \tag{4.3}$$

The model can simulate the end to end delay which implies that the time in queue can be calculated from Equation 4.3. Simulation Parameters for the topology is shown in Table 4.4 and the network activity is based on uniform distribution of time when the data is sent, from a random source to a random destination with random data size. In this implementation there is a lot of traffic in the network and all data is sent within the HB interval and the reset buffers are cleared for every cycle.

Parameters	Unit
Clock Frequency	20 MHz
Channel Delay	0.1 µs
Data Rate	50 Mbps
HeartBeat Interval	25 µs
Samples Taken	26
Simulation Time	0.000 65 s

Table 4.4: Simulation Parameters for Generic Random Topology

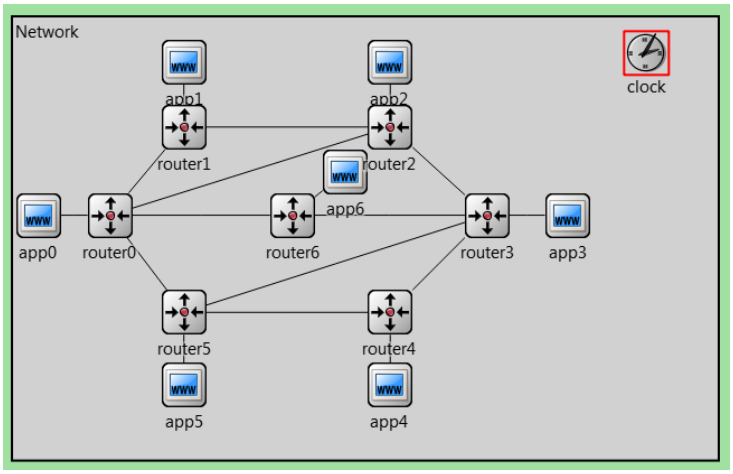


Figure 4.7: Generic Random Topology Representation

4.2.1 Generic Random Topology Results Analysis

Similar to the previous case study the results are analysed by comparing static routing with dynamic routing. Figure 4.8 shows the raw values for the static routing. In this graph the max value is around 265 bytes and the effect of resetting the buffer can be clearly seen. Figure 4.9 shows a similar pattern in dynamic routing where a new route is selected for every cycle of HB and the max value is around the same but at a different simulation time. Note that the Y Axis scales are different in both the graphs.

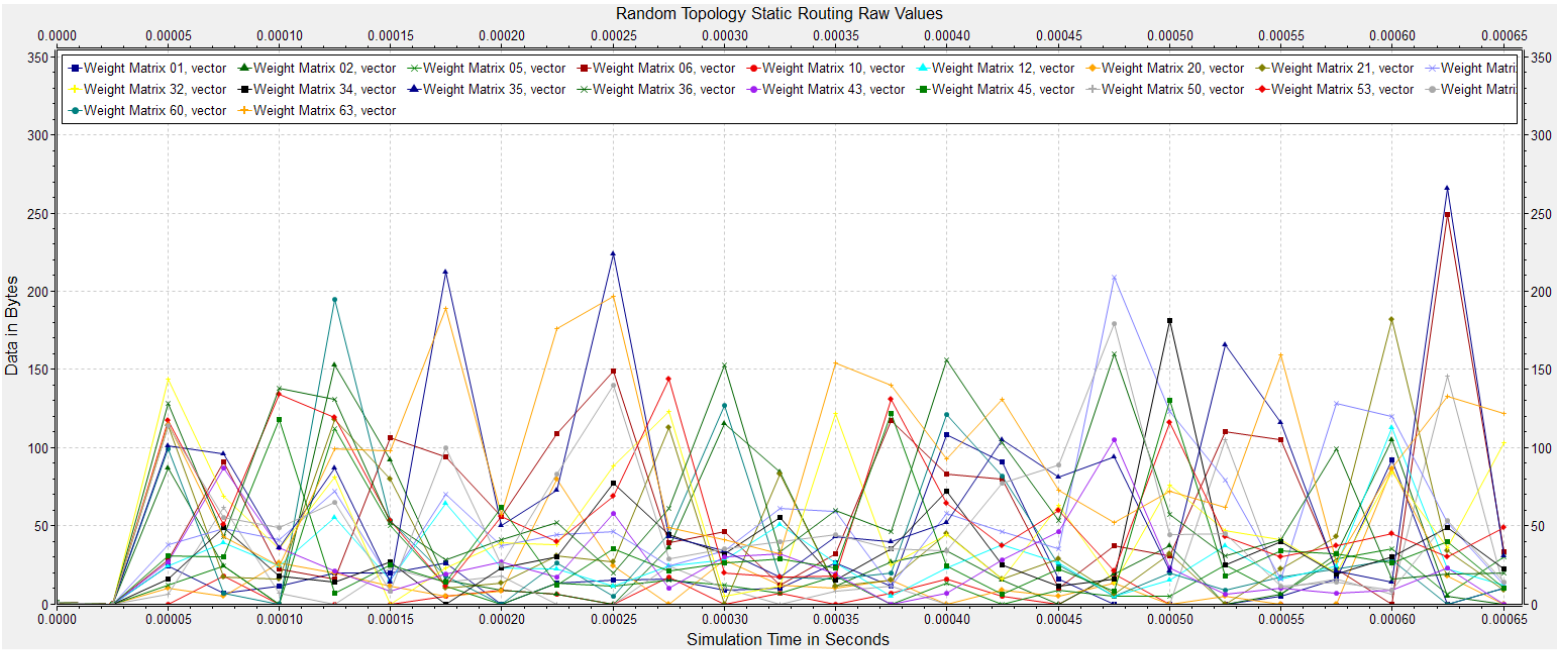


Figure 4.8: Random Topology Static Routing Raw Values

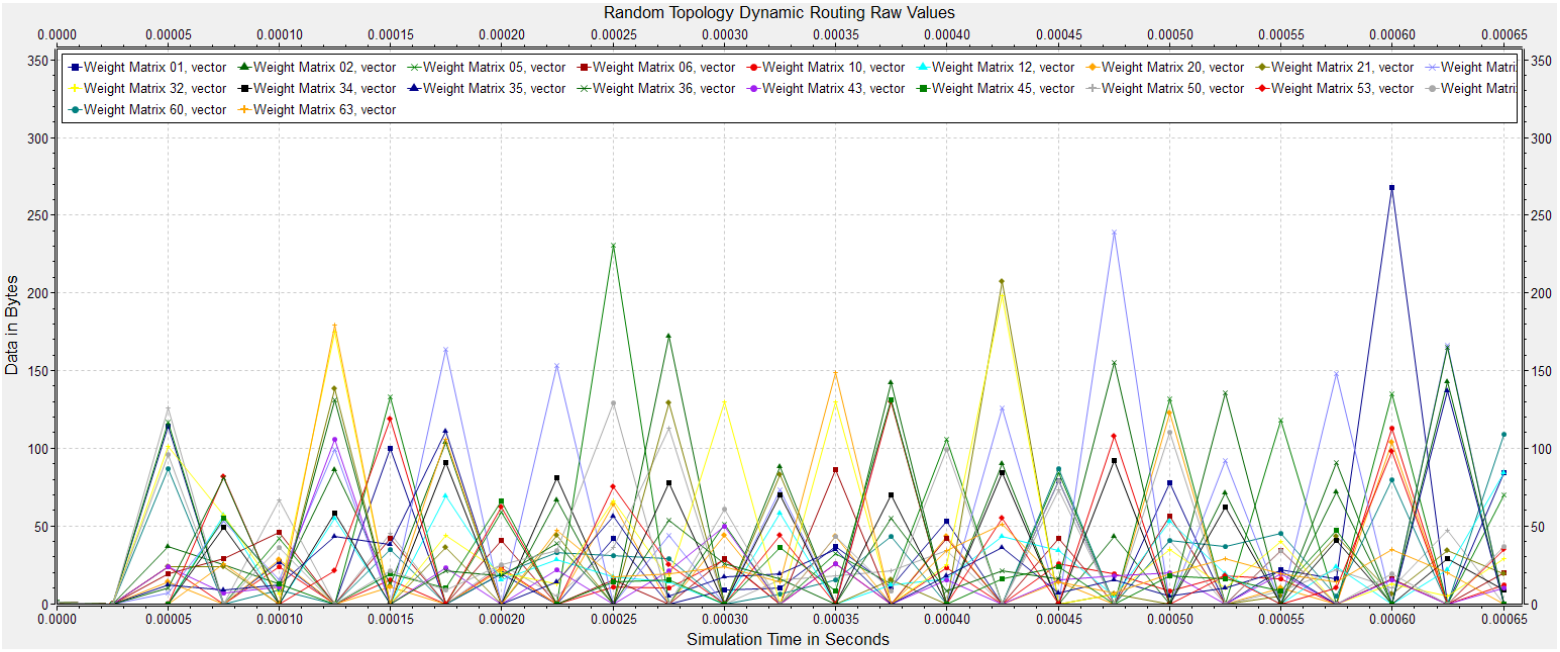


Figure 4.9: Random Topology Dynamic Routing Raw Values

Figure 4.10 shows the mean values of data in the links in static routing and Figure 4.11 shows them for dynamic routing. This comparison clearly shows that the minimum value is around 5 bytes and maximum around 90 bytes while range for dynamic routing is between 15 bytes

and 58 bytes. This shows that the dynamic routing is much effective when there is more data on the network and when it is random.Note that the Y Axis scales are different in both the graphs.

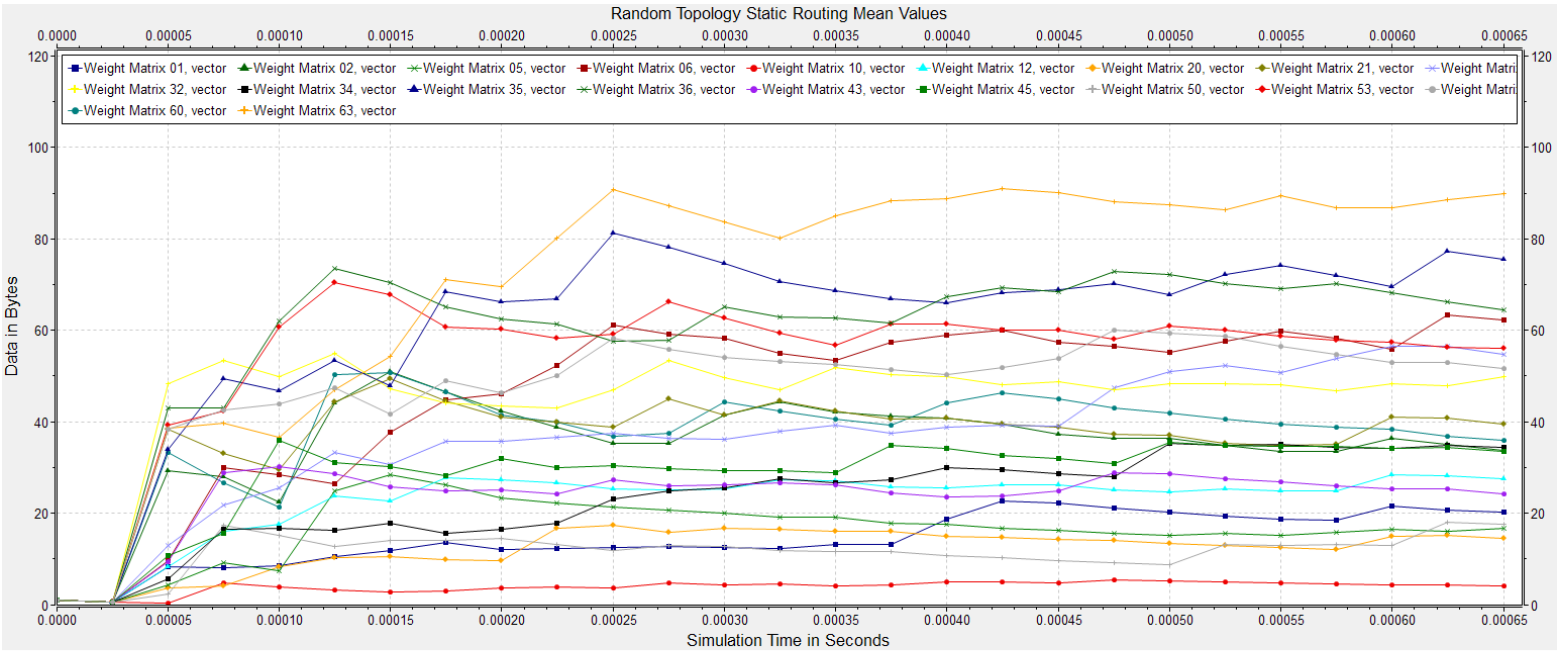


Figure 4.10: Random Topology Static Routing Mean Values

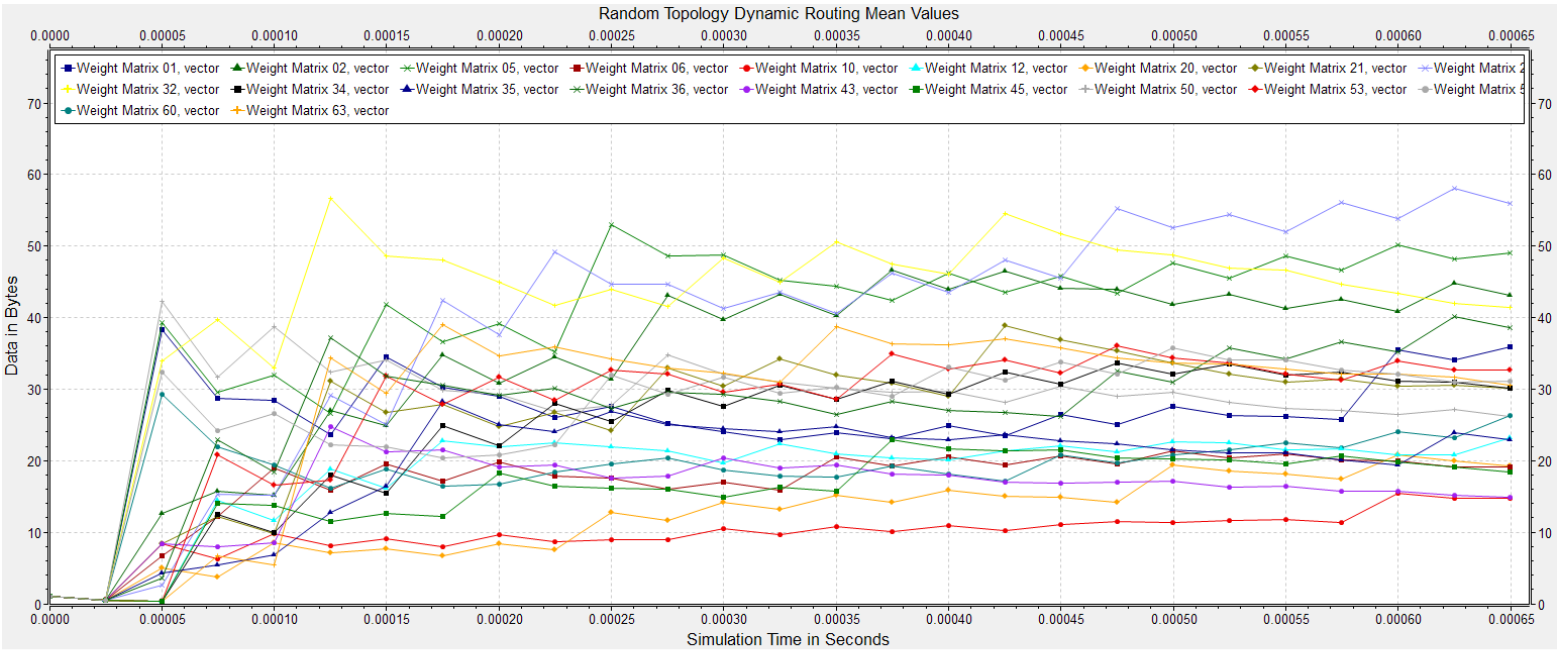


Figure 4.11: Random Topology Dynamic Routing Mean Values

Figure 4.12 shows total sum of data in each link for static routing and Figure 4.13 shows for

dynamic routing. Again the minimum value for static routing is around 100 bytes and maximum around 2500 bytes while range for dynamic routing is between 400 bytes and 1600 bytes. Note that the Y Axis scales are different in both the graphs.

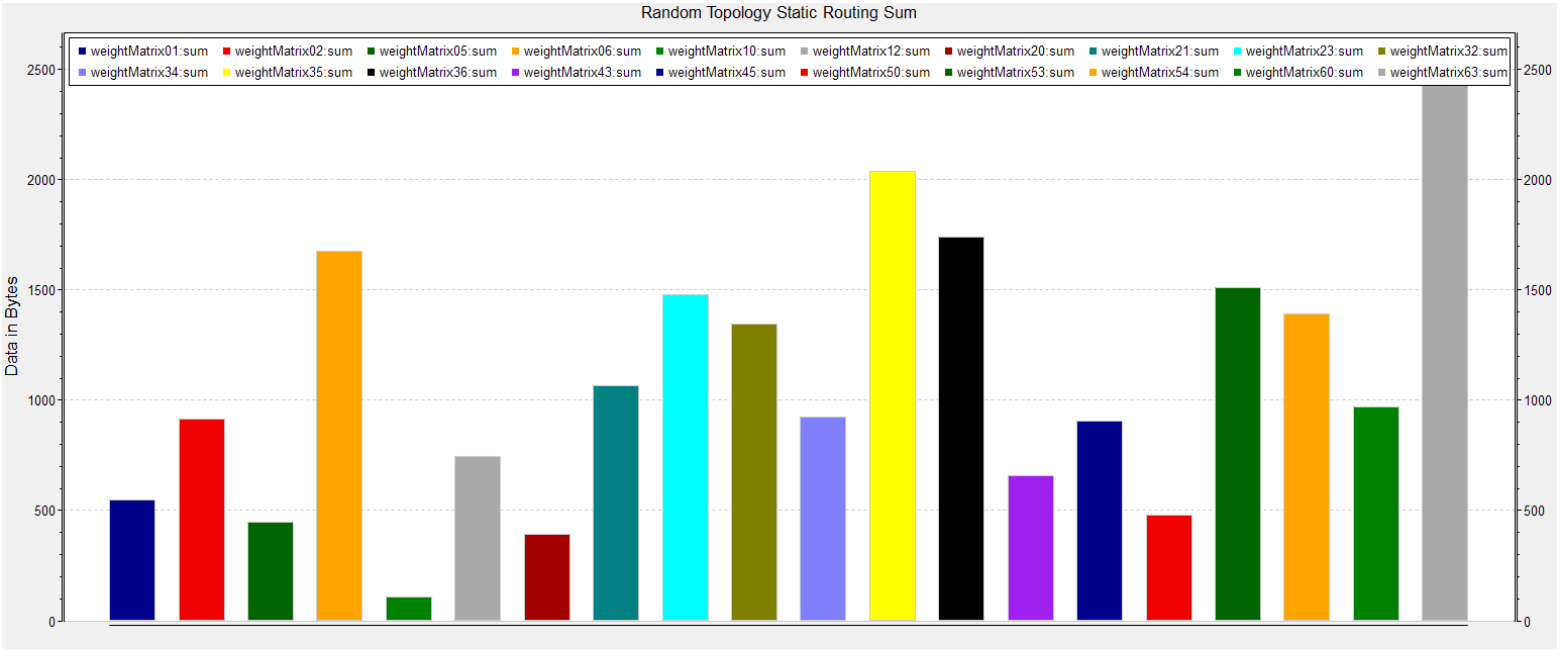


Figure 4.12: Random Topology Static Routing Sum of Data in Each Link

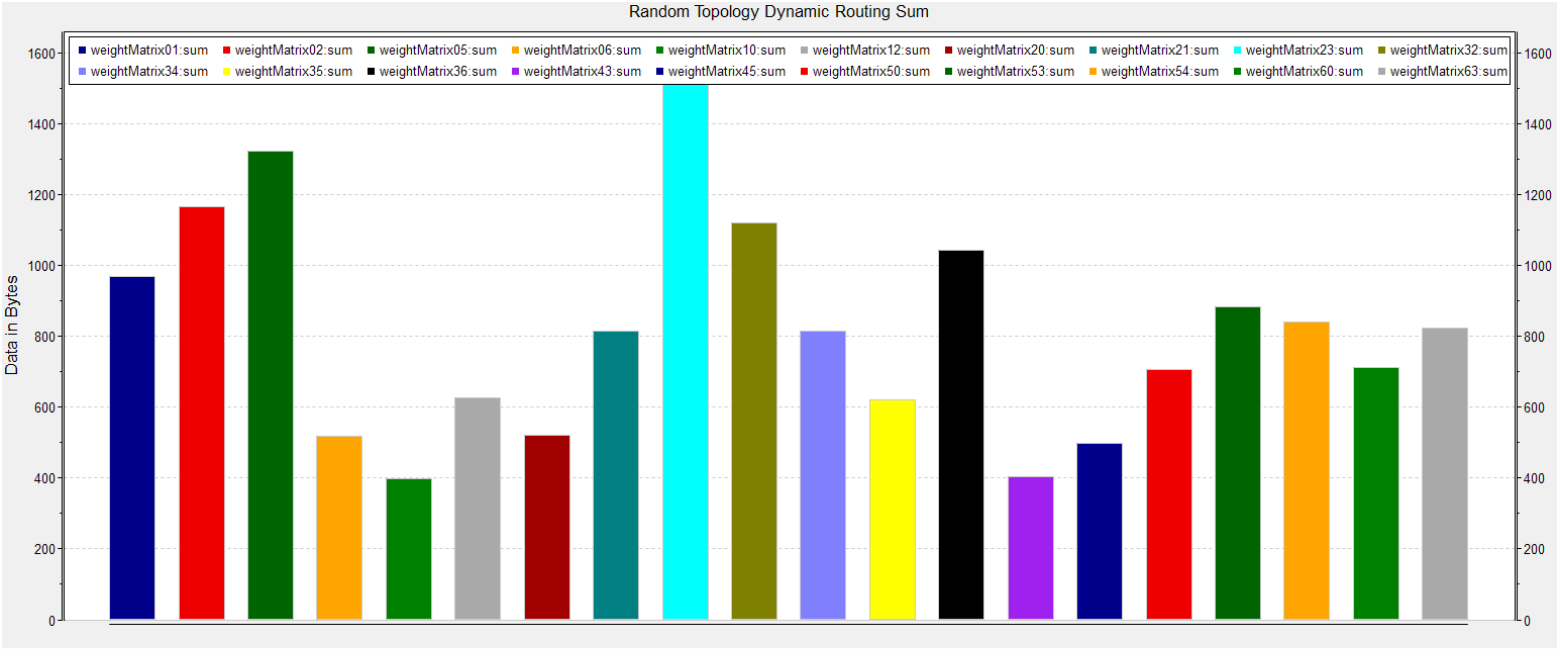


Figure 4.13: Random Topology Dynamic Routing Sum of Data in Each Link

Figure 4.14 shows random data sent from Node 0 to random destinations over the simulation time. The end to delay from Node 0 to other destinations for static routing is shown in Figure 4.15 and Figure 4.16 shows the end to end delay for dynamic routing. The comparisons clearly shows that the dynamic routing has less end to end delay, the maximum delay is reduced by at least 40% and and the minimum delay is reduced by a factor of 2. The peaks observed in the graphs occur precisely when HB are sent when there is more data on the network. Similarly the end to end delays from other nodes to random destinations are shown in the Appendix A. They are self explanatory and show a very similar result to Node 0. Note that the Y Axis scales are different in both the graphs.

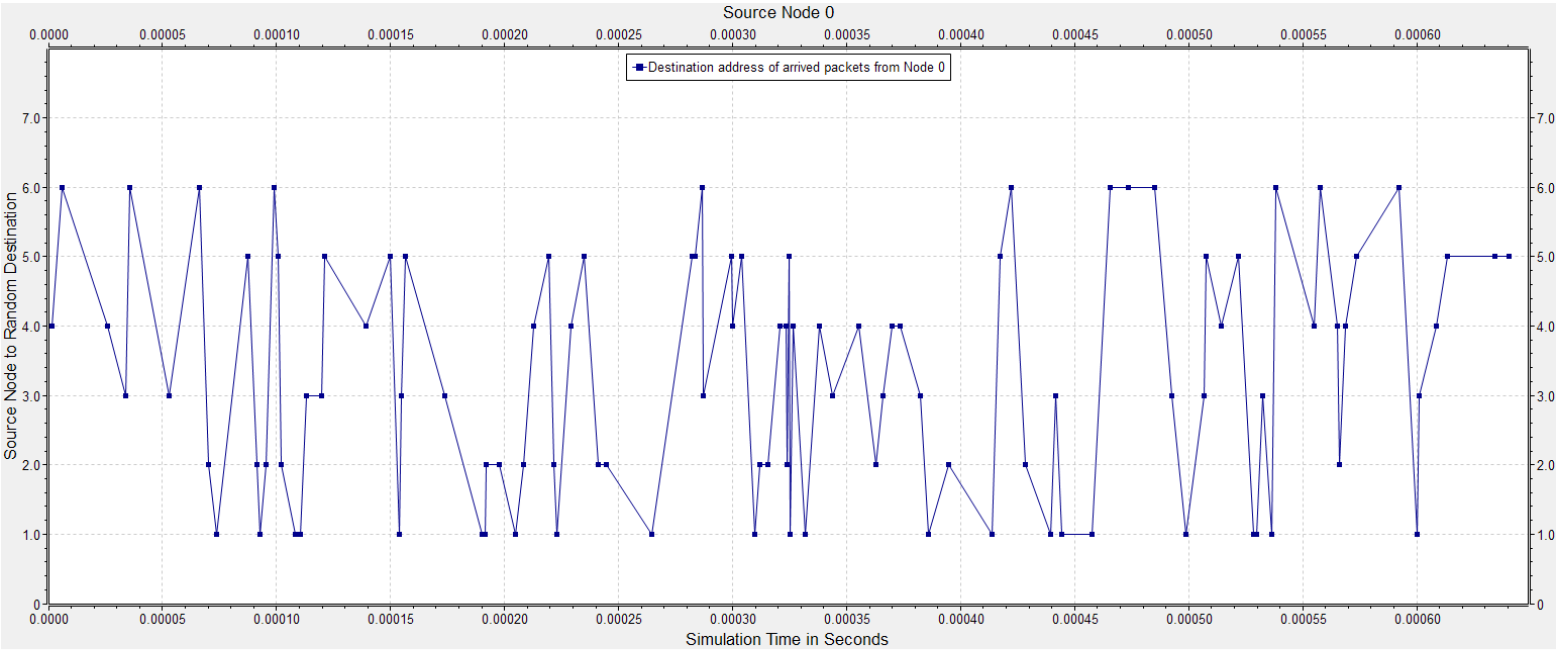


Figure 4.14: Random Topology Data Sent from Node 0 to Random Destination

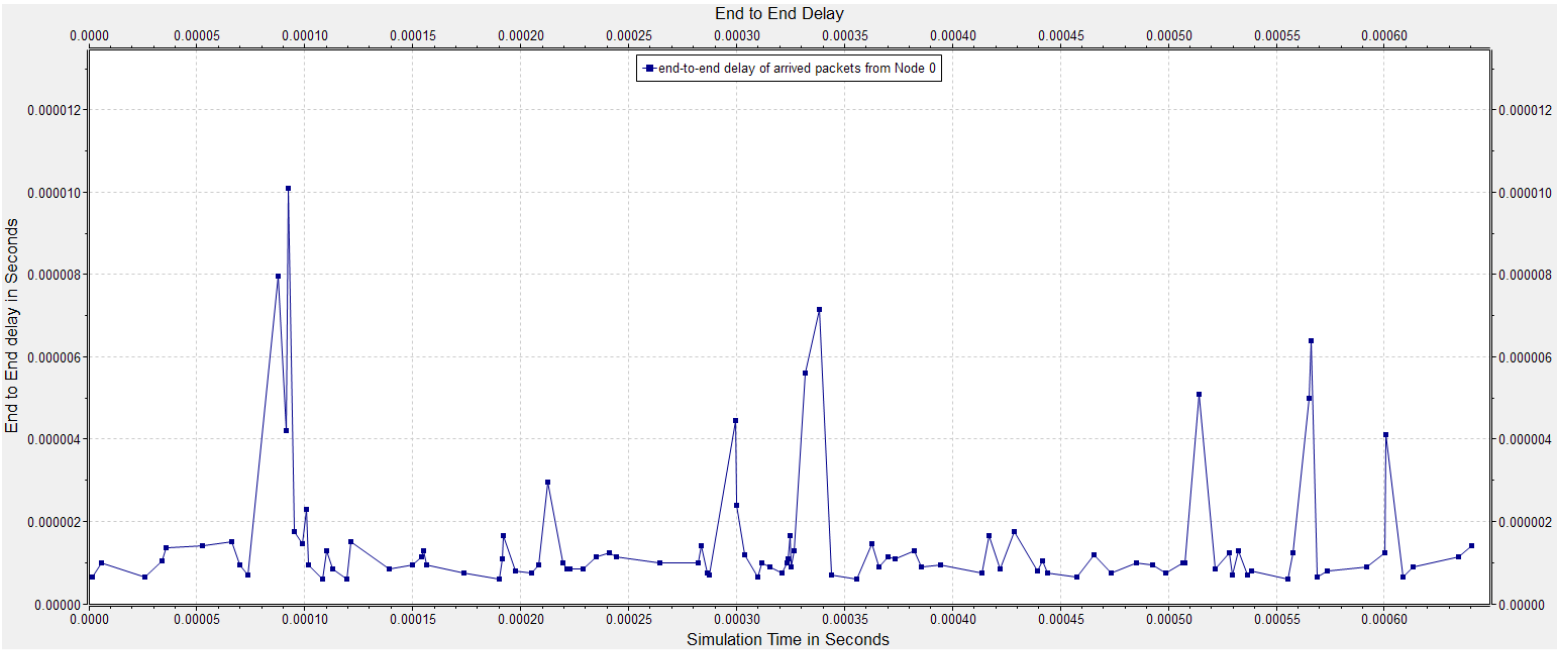


Figure 4.15: Random Topology Static Routing End To End Delay of Packets from Node 0

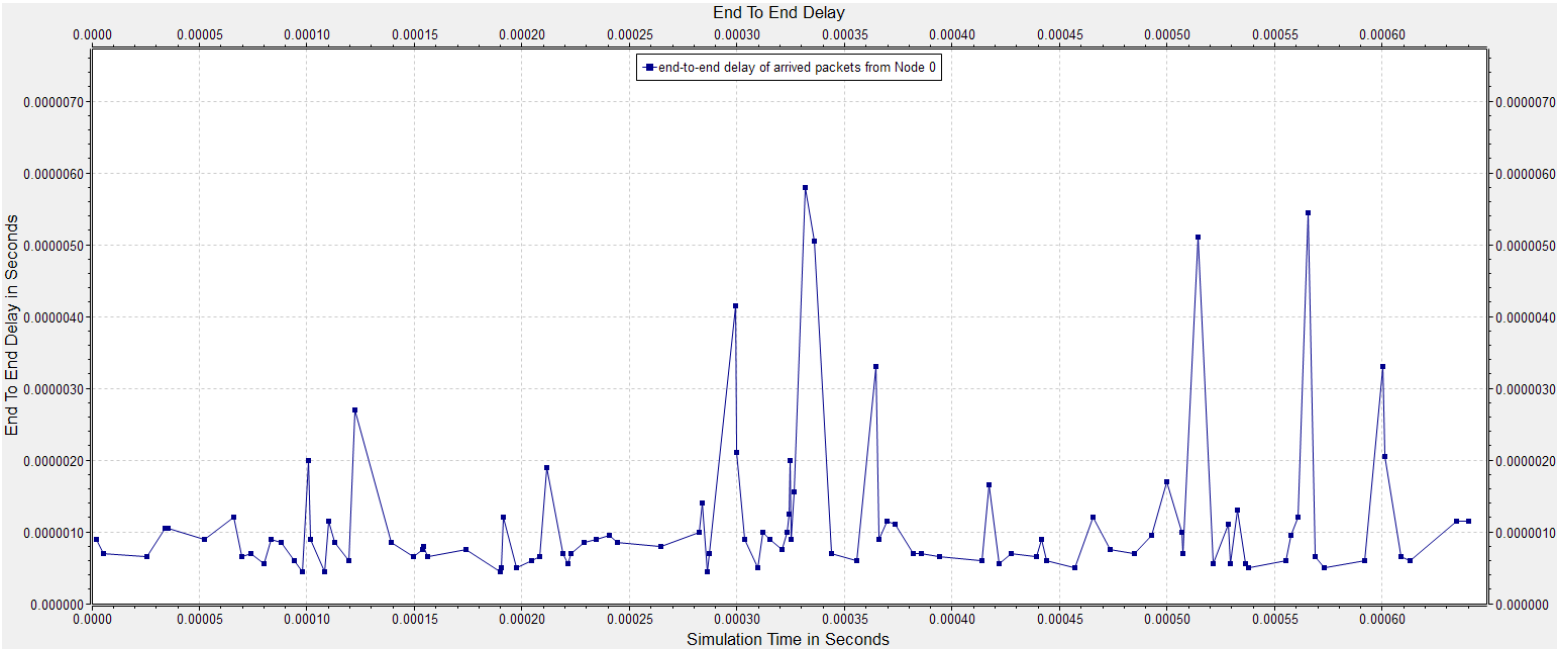


Figure 4.16: Random Topology Dynamic Routing End To End Delay of Packets from Node 0

Preliminary results based on the end to end delay for $26\,\mathrm{samples}$ and a duration of $0.000\,65\,\mathrm{s}$ shows that the routing algorithm proposed works effectively when there is more data on the network and can be significantly improved by adding more parameters such as end to end delay and time in queue for the optimal route computation.

Chapter 5

Conclusions

This report shows that it is possible to develop a new routing algorithm for a distributed On Board Computer which exclusively works in a space environment. The ATON case study shows that the routing algorithm may not be effective in a smaller topology with one to one mapping of nodes but on the contrary a generic topology with more nodes and more traffic, the algorithm is very effective in routing data as well as optimizing the link usage. The preliminary results for end to end delay measured in the simulation shows that the dynamic routing algorithm reduces the delay significantly. Although one significant disadvantage with the algorithm is that the worker nodes have the updated weight matrix with a delay of one HeartBeat cycle.

The score computation can be improved and further optimized by adding more parameters such as end to end delay, dropped packets and time in the queue to find a better route. Another option will be to increase the frequency of the HeartBeat interval but this can increase the overhead of the network traffic. This routing algorithm is very flexible and can be implemented easily and is scalable without changing the SpaceWire IPC and the reconfiguration graph. The algorithm should also be evaluated for more samples of data and also by increasing the number of nodes. Finally to conclude, a new dynamic routing algorithm is proposed, implemented and evaluated in simulation which offers a deterministic way of routing data using a PULL type Heartbeat monitoring mechanism.

5.1 Lessons Learned

During the wormhole switching implementation, the queues were allowing flits to pass through even when their status was blocked, upon debugging the root cause was found to be that multiple nodes were trying to access the same queue creating a race condition. This was fixed by introducing unique flit ids and mutual exclusion to block the queues based on their order of arrival.

Chapter 6

Future Work

A new decentralized approach using a PUSH transmissions is proposed, in this approach the weight matrix is shared only with the local neighbours at regular intervals instead of a Heart-Beat approach. This will be a modified version of research and development proposed in [14] where the OBC is based on a CANBus and works on broadcasting.

Figure 6.1 shows the weight matrix of the generic random topology implemented in the second use case, with just 4 PUSH messages all nodes will have the current status of the network.

```
int topology[7][7] =  { {0, 1, 1, 0, 0, 1, 1},
                        {1, 0, 1, 0, 0, 0, 0},
                        {1, 1, 0, 1, 0, 0, 0},
                        {0, 0, 1, 0, 1, 1, 1},
                        {0, 0, 0, 1, 0, 1, 0},
                        {1, 0, 0, 1, 1, 0, 0},
                        {1, 0, 0, 1, 0, 0, 0},
                      };
```

Figure 6.1: Weight Matrix of the Generic Random Topology

There are many advantages with this approach and they summarized below:

- It offers decentralized control which means all nodes will have management functionalities and the roles of master and observer will no longer be valid while the reconfiguration will still be possible with priority addressing.
- All nodes will have the current weight matrix before taking the routing decision.
- Scalable to even larger networks and deadlocks caused by health monitoring messages is completely eliminated.
- Can be easily implemented and maintained.

By increasing the frequency of the transmissions the network information is available at a quicker rate which implies better routing decisions but can significantly increase the network

overhead which is a trade-off to be considered in the system design.

Finally both the PUSH and PULL type mechanisms can be implemented in the ScOSA hardware and can be evaluated further.

Bibliography

[1] Jens Eickhoff. *Onboard Computers, Onboard Software and Satellite Operations An Introduction*. Springer, 2012.

[2] NASA. *James Webb Space telescope*. (accessed September 20, 2019). URL: https://www.jwst.nasa.gov/.

[3] ESA. *Philae Lander*. (accessed September 20, 2019). URL: https://rosetta.esa.int/.

[4] Daniel P. Siewiorek and Priya Narasimhan. "FAULT-TOLERANT ARCHITECTURES FOR SPACE AND AVIONICS APPLICATIONS". In: 2005.

[5] Cobham Gaisler. *Leon 4 Fault Tolerant Procssor*. (accessed September 20, 2019). URL: https://www.gaisler.com/index.php/products/processors/leon4ft.

[6] Carl Treudler et al. "ScOSA - Scalable On-Board Computing for Space Avionics". In: Oct. 2018.

[7] T. Peng et al. "A new SpaceWire protocol for reconfigurable distributed on-board computers: SpaceWire networks and protocols, long paper". In: *2016 International SpaceWire Conference (SpaceWire)*. 2016, pp. 1–8. DOI: 10.1109/SpaceWire.2016.7771624.

[8] D. Lüdtke et al. "OBC-NG: Towards a reconfigurable on-board computing architecture for spacecraft". In: *2014 IEEE Aerospace Conference*. 2014, pp. 1–13. DOI: 10.1109/AERO.2014.6836179.

[9] Leon Alkalai. "An overview of flight computer technologies for future NASA space exploration missions". In: *Acta Astronautica - ACTA ASTRONAUT* 52 (May 2003), pp. 857–867. DOI: 10.1016/S0094-5765(03)00066-3.

[10] R. Ginosar. "Survey of Processors for Space". In: *DASIA 2012 - DAta Systems In Aerospace*. Vol. 701. ESA Special Publication. Aug. 2012, p. 10.

[11] P. Behr et al. "Fault Tolerance and COTS: Next generation of high performance satellite computers". In: Jan. 2003.

[12] Carlos Villalpando et al. "Reliable multicore processors for NASA space missions". In: Apr. 2011, pp. 1–12. DOI: 10.1109/AERO.2011.5747447.

[13] Paul Murray et al. “High performance, high volume reconfigurable processor architecture”. In: (Mar. 2012). DOI: 10.1109/AERO.2012.6187234.

[14] H. Yashiro et al. “A high assurance space on-board distributed computer system”. In: *2003 IEEE Aerospace Conference Proceedings (Cat. No.03TH8652)*. Vol. 5. 2003. DOI: 10.1109/AERO.2003.1235175.

[15] J. Samson et al. “Technology Validation: NMP ST8 Dependable Multiprocessor Project II”. In: *2007 IEEE Aerospace Conference*. 2007, pp. 1–18. DOI: 10.1109/AERO.2007.352784.

[16] John Samson. “Update on Dependable Multiprocessor CubeSat technology development”. In: (Mar. 2012). DOI: 10.1109/AERO.2012.6187238.

[17] Ting Peng Kurt Schwenk Moritz Ulmer. “ScOSA: application development for a high-performance space qualified onboard computing platform”. In: 2018.

[18] A. Vinogradov, E. Yablokov, and V. Yachnaya. “Upgrade of Ethernet-SpaceWire Protocol”. In: *2019 24th Conference of Open Innovations Association (FRUCT)*. 2019, pp. 486–492. DOI: 10.23919/FRUCT.2019.8711937.

[19] Tanyalak Vatinvises. “Monitoring Mechanism Design for a Distributed Reconfigurable Onboard Computer”. In: 2016.

[20] K. Höflinger et al. “Dynamic Fault Tree Analysis for a Distributed Onboard Computer”. In: *2019 IEEE Aerospace Conference*. 2019, pp. 1–13. DOI: 10.1109/AERO.2019.8742128.

[21] “IEEE Standard for Heterogeneous InterConnect (HIC), (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)”. In: *IEEE Std 1355-1995* (June 1996), pp. 1–148. DOI: 10.1109/IEEESTD.1996.81004.

[22] “ECSS-E-ST-50-12C Rev.1 – SpaceWire – Links, nodes, routers and networks”. In: (2019).

[23] Steve Parkes. “SpaceWire User’s Guide”. In: (2012).

[24] D. Yi et al. “SpaceWire standard and improved wormhole router design”. In: *2012 IEEE Aerospace Conference*. Mar. 2012, pp. 1–8. DOI: 10.1109/AERO.2012.6187103.

[25] Ville Rantala, Teijo Lehtonen, and Juha Plosila. “Network on Chip Routing Algorithms”. In: (Dec. 2008).

[26] Jianxiao Zou, Zheng-qian Zhang, and Hong-bing Xu. “Design of heartbeat invalidation detecting mechanism in triple modular redundancy multi-machine system”. In: *COMPEL: The International Journal for Computation and Mathematics in Electrical and Electronic Engineering* 29 (Mar. 2010), pp. 495–504. DOI: 10.1108/03321641011014959.

[27] Hermann Kopetz. *Real-Time Systems Design Principles for Distributed Embedded Applications*. Springer, 2011.

[28] Salah Abdulghani, Sara Hazim, and Abdullah Mohammed Salih. “Performance Evaluation and Comparison of Dynamic Routing Protocols for Suitability and Reliability”. In: *International Journal of Grid and Distributed Computing* 11 (July 2018), pp. 41–52. DOI: 10.14257/ijgdc.2018.11.7.05.

[29] Sabina Barakovic, Suad Kasapović, and Jasmina Barakovic Husic. “Comparison of MANET Routing Protocols in Different Traffic and Mobility Models”. In: *TELFOR Journal* 2 (June 2010), pp. 8–12.

[30] Cui-Qing Yang and A. V. S. Reddy. “A taxonomy for congestion control algorithms in packet switching networks”. In: *IEEE Network* 9.4 (July 1995), pp. 34–45. DOI: 10.1109/65.397042.

[31] M. Sivabalan and H. T. Mouftah. “On the Design of Link-State Routing Protocol for Connection-Oriented Networks”. In: *Journal of Network and Systems Management* 9.2 (June 2001), pp. 223–242. ISSN: 1573-7705. DOI: 10.1023/A:1011319226517. URL: https://doi.org/10.1023/A:1011319226517.

[32] Andrii Kovalov et al. “Task-Node Mapping in an Arbitrary Computer Network Using SMT Solver”. In: Sept. 2017, pp. 177–191. ISBN: 978-3-319-66844-4. DOI: 10.1007/978-3-319-66845-1_12.

[33] Varga A. “OMNeT++: Modeling and Tools for Network Simulation”. In: (2010). https://www.omnetpp.org.

[34] Stephan Theil et al. “ATON - Autonomous Terrain-based Optical Navigation for Exploration Missions: Recent Flight Test Results”. In: Sept. 2017.

[35] FRABOUL Christian FERRANDIZ Thomas FRANCES Fabrice. “Worst-case end-to-end delays evaluation for SpaceWire networks”. In: *Event Dynamic Systems, 2011, vol. 21, n° 3, pp. 339-357* (2011).

[36] K. Borchers et al. “Time-triggered data transfers over SpaceWire for distributed systems”. In: *2018 IEEE Aerospace Conference*. Mar. 2018, pp. 1–11. DOI: 10.1109/AERO.2018.8396435.

Appendix A

Random Topology Results

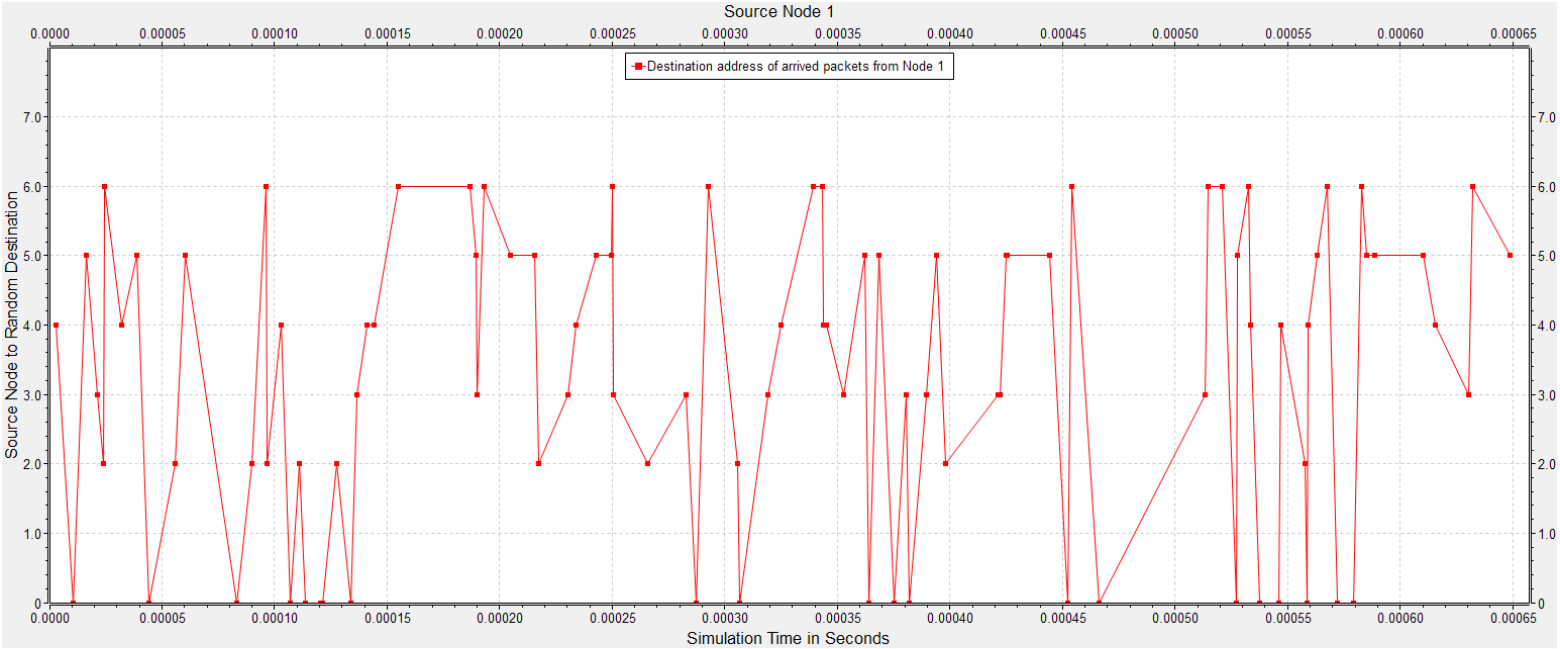


Figure A.1: Random Topology Data Sent from Node 1 to Random Destination

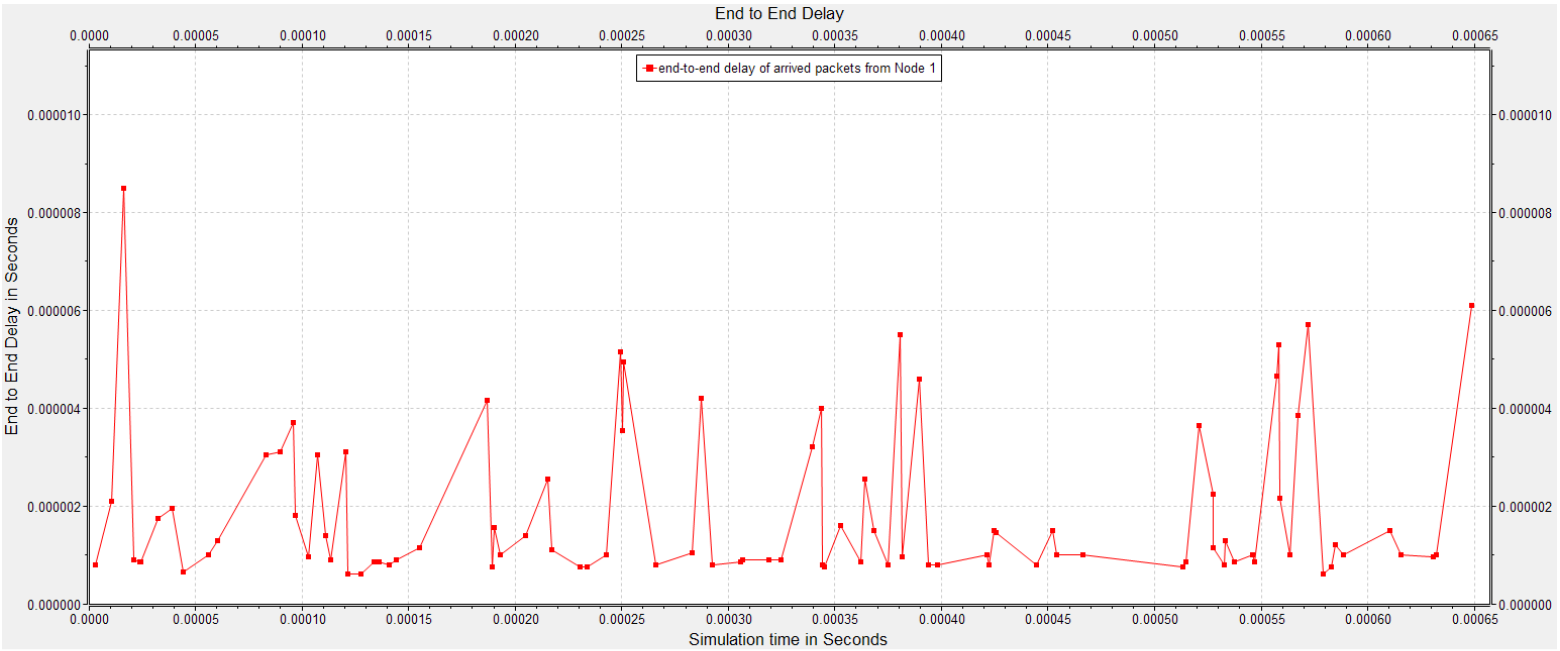


Figure A.2: Random Topology Static Routing End To End Delay of Packets from Node 1

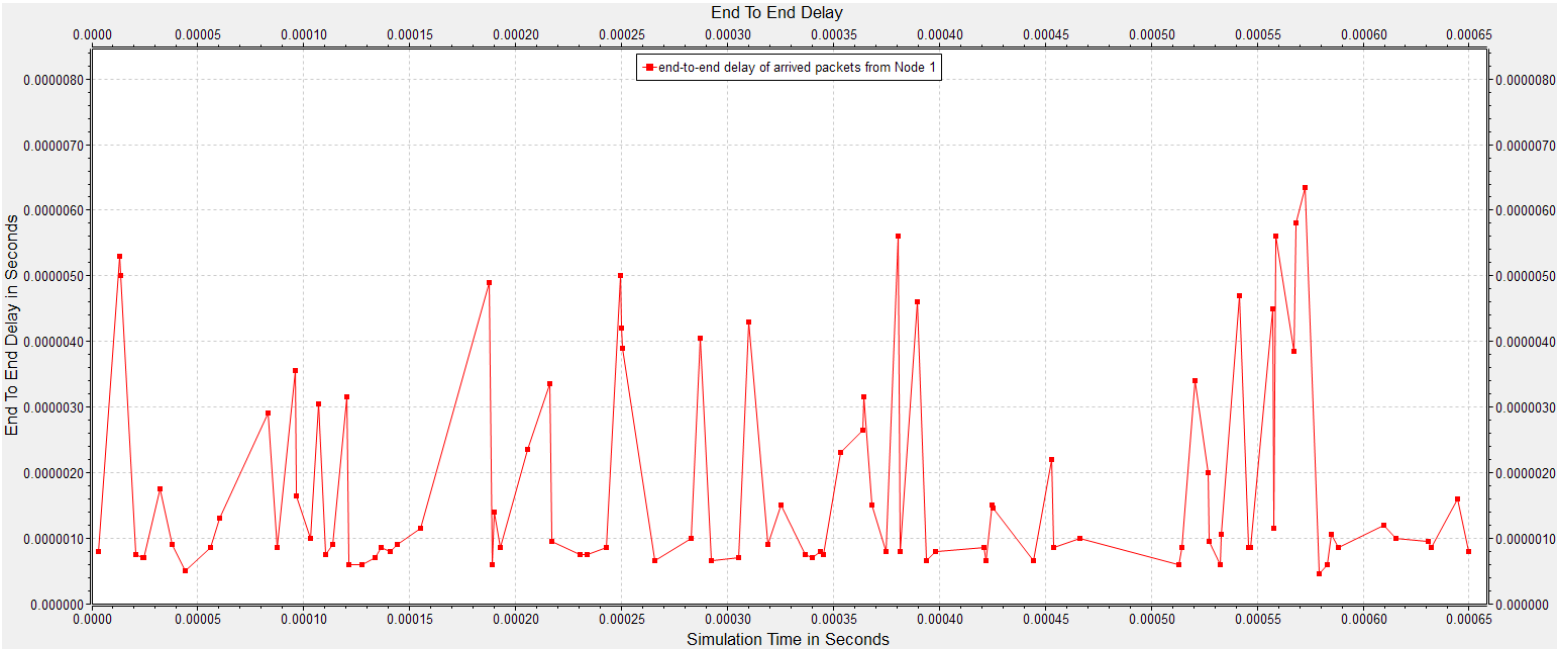


Figure A.3: Random Topology Dynamic Routing End To End Delay of Packets from Node 1

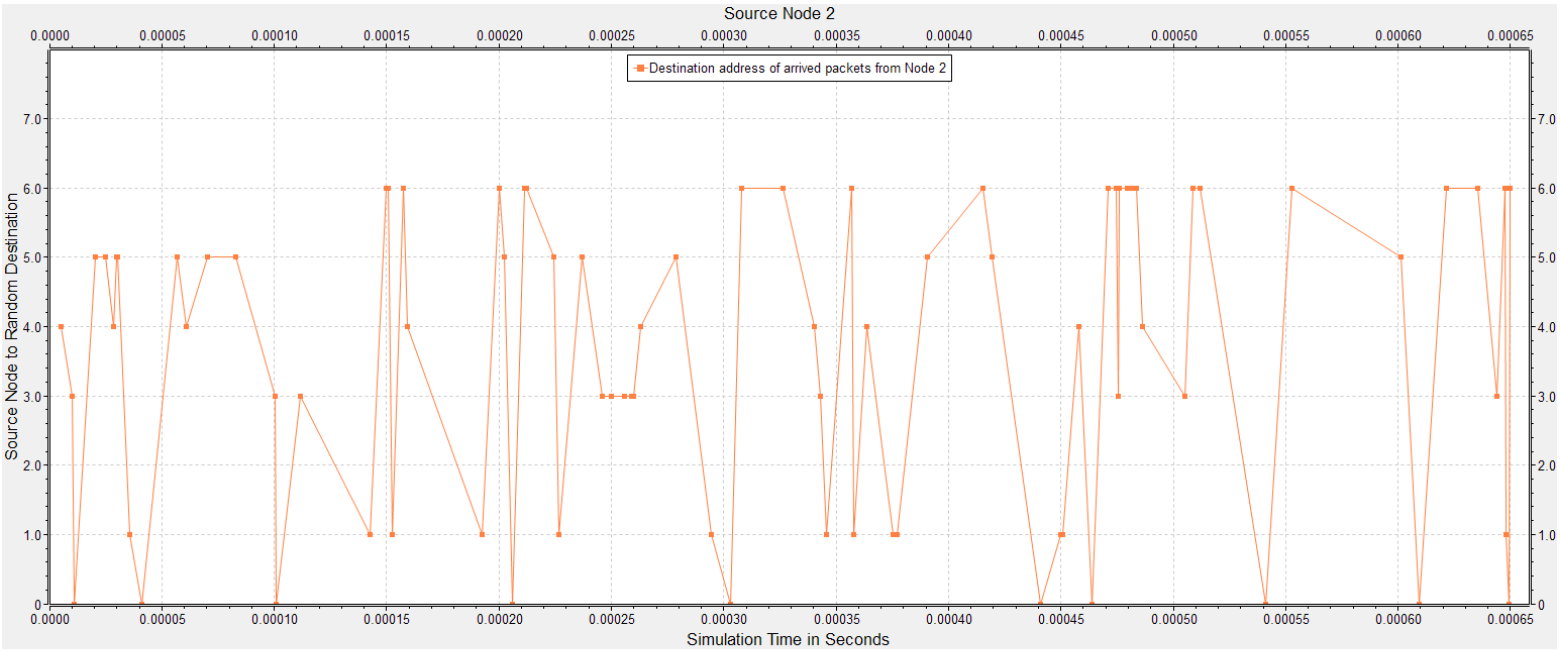


Figure A.4: Random Topology Data Sent from Node 2 to Random Destination

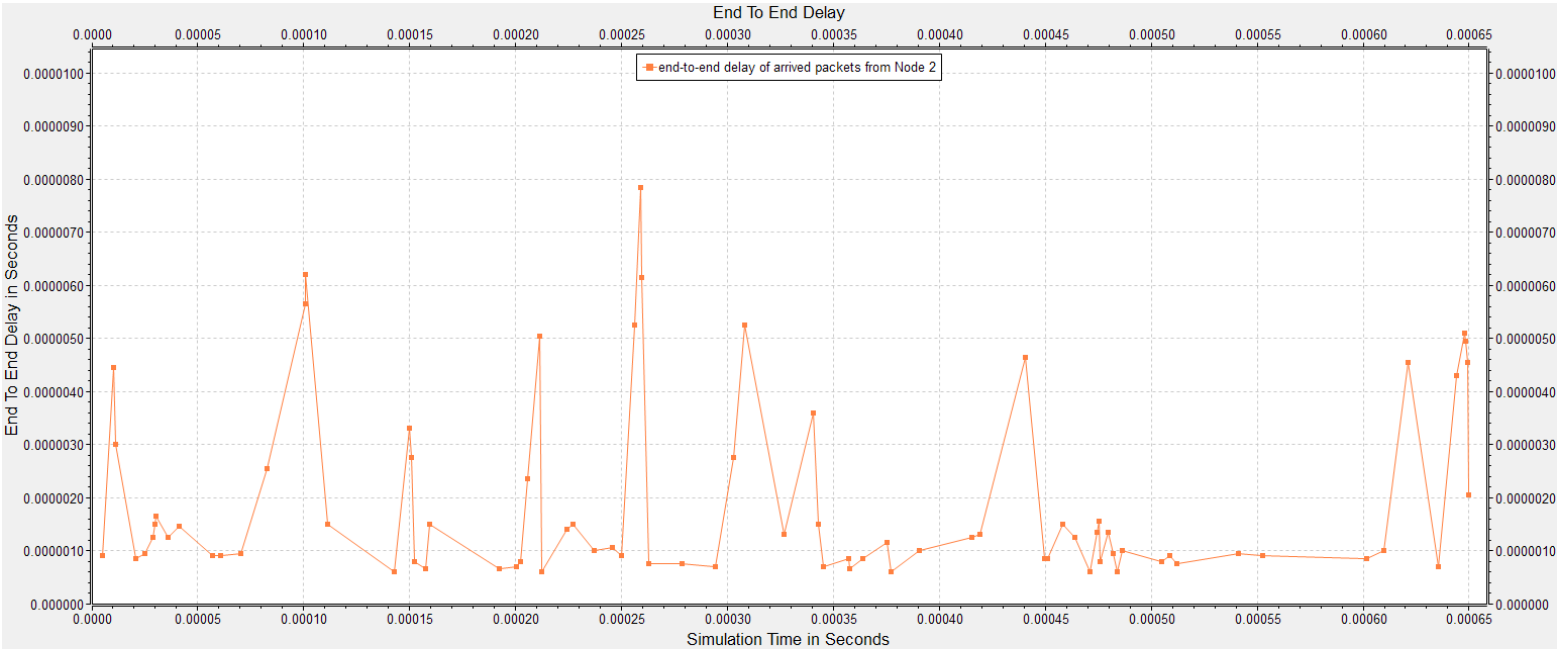


Figure A.5: Random Topology Static Routing End To End Delay of Packets from Node 2

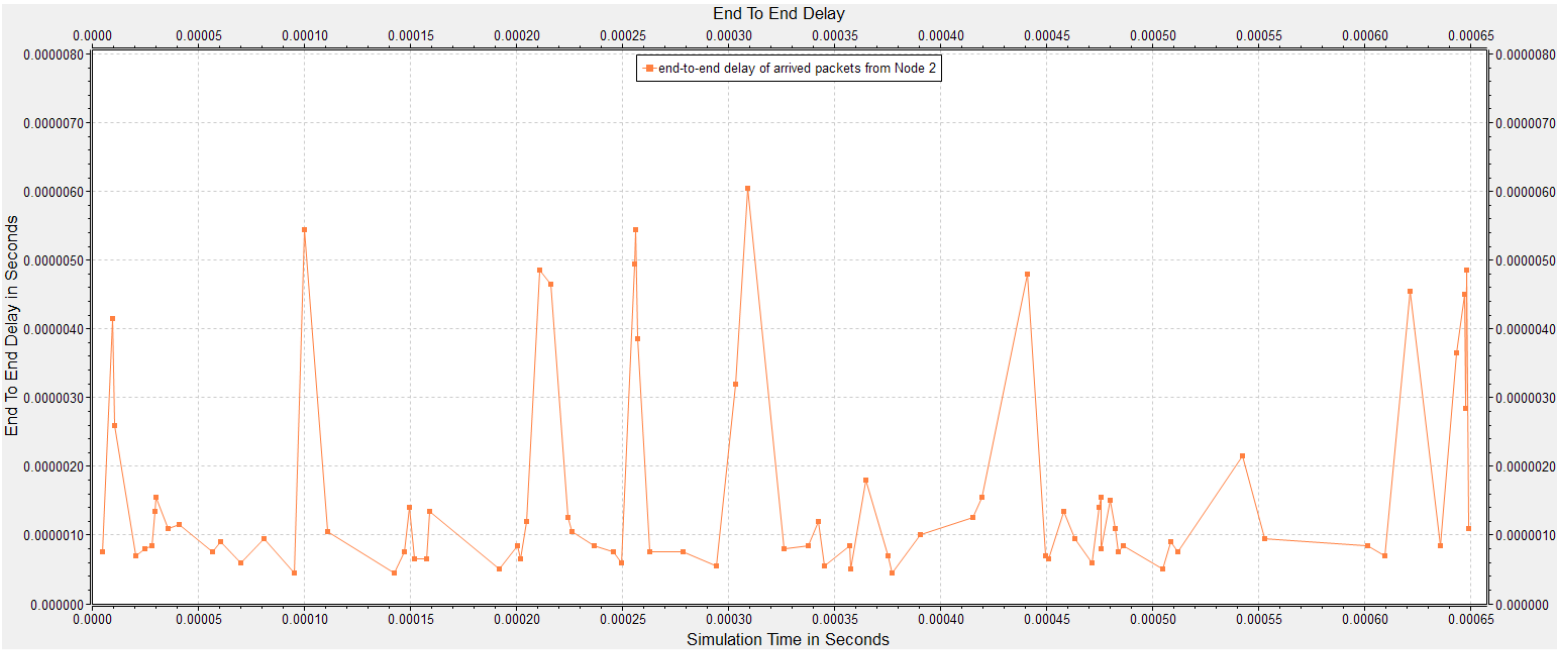


Figure A.6: Random Topology Dynamic Routing End To End Delay of Packets from Node 2

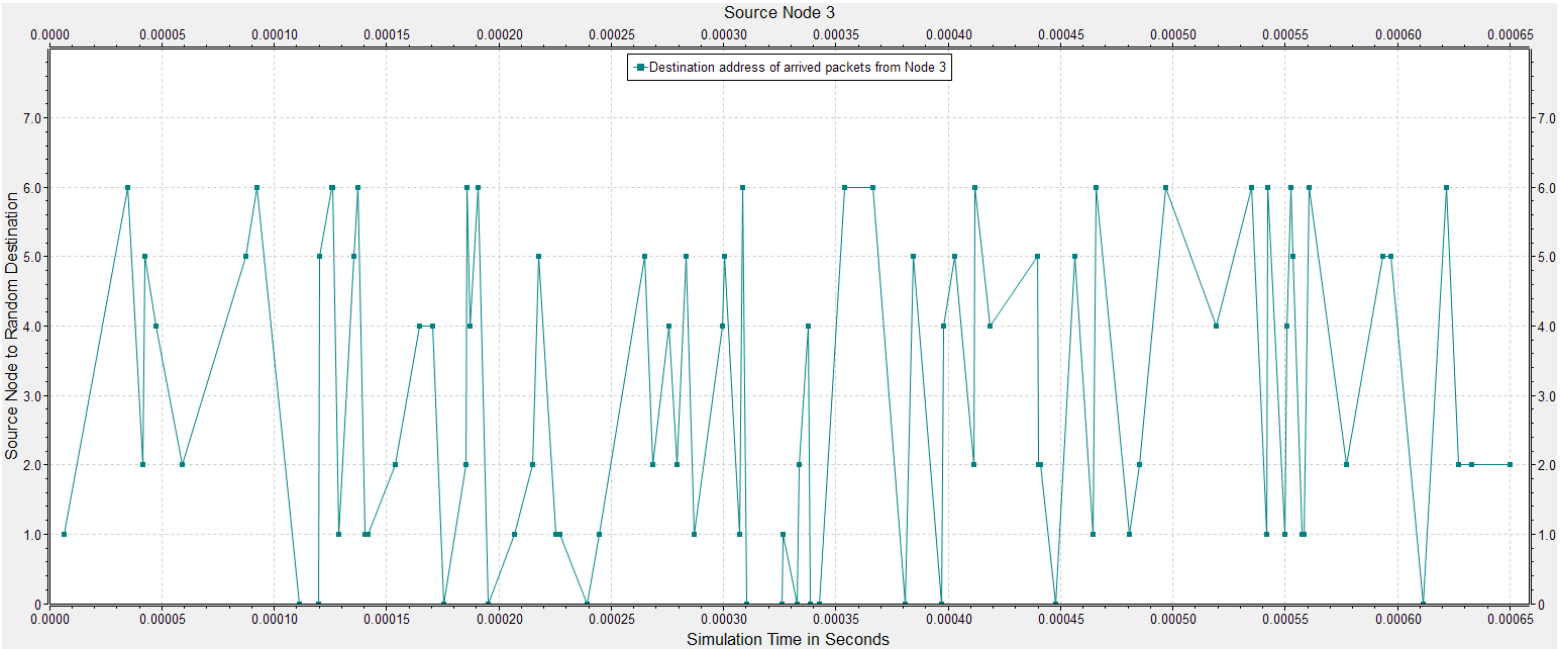


Figure A.7: Random Topology Data Sent from Node 3 to Random Destination

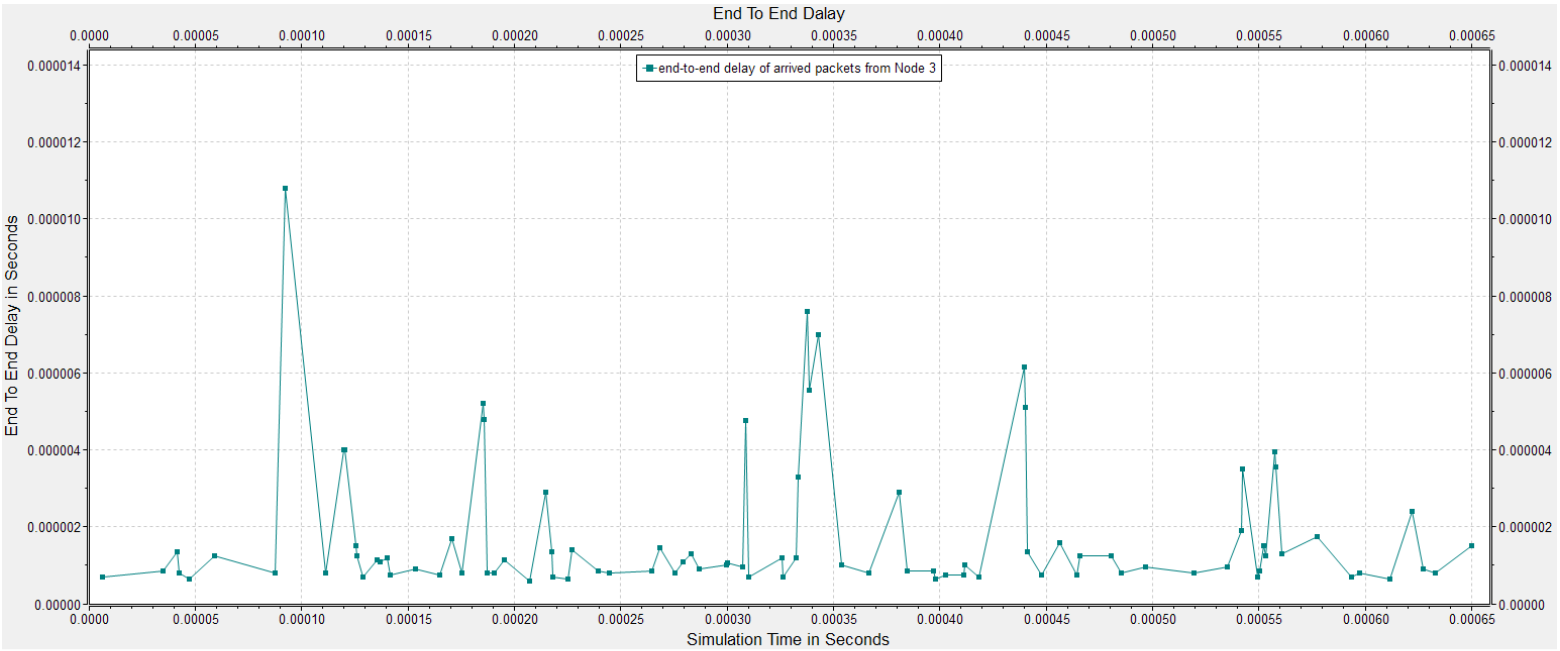


Figure A.8: Random Topology Static Routing End To End Delay of Packets from Node 3

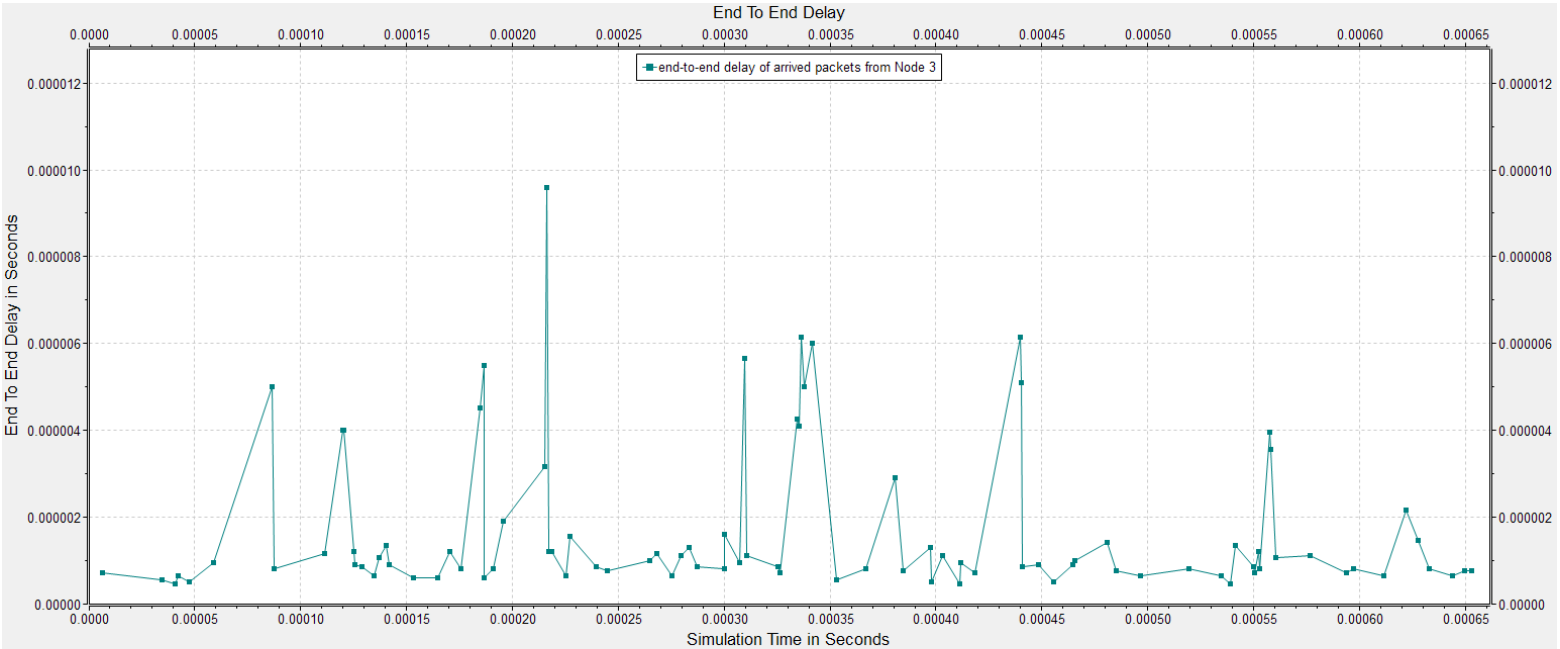


Figure A.9: Random Topology Dynamic Routing End To End Delay of Packets from Node 3

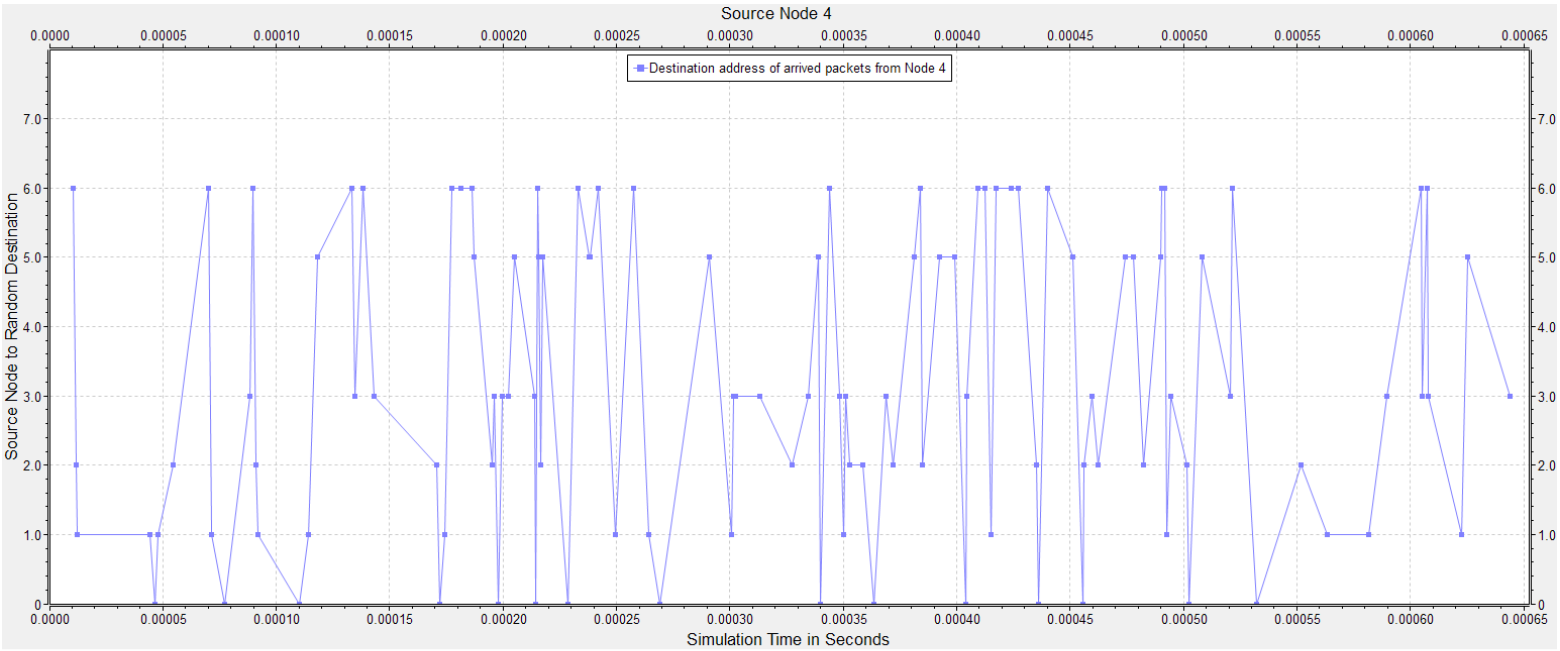


Figure A.10: Random Topology Data Sent from Node 4 to Random Destination

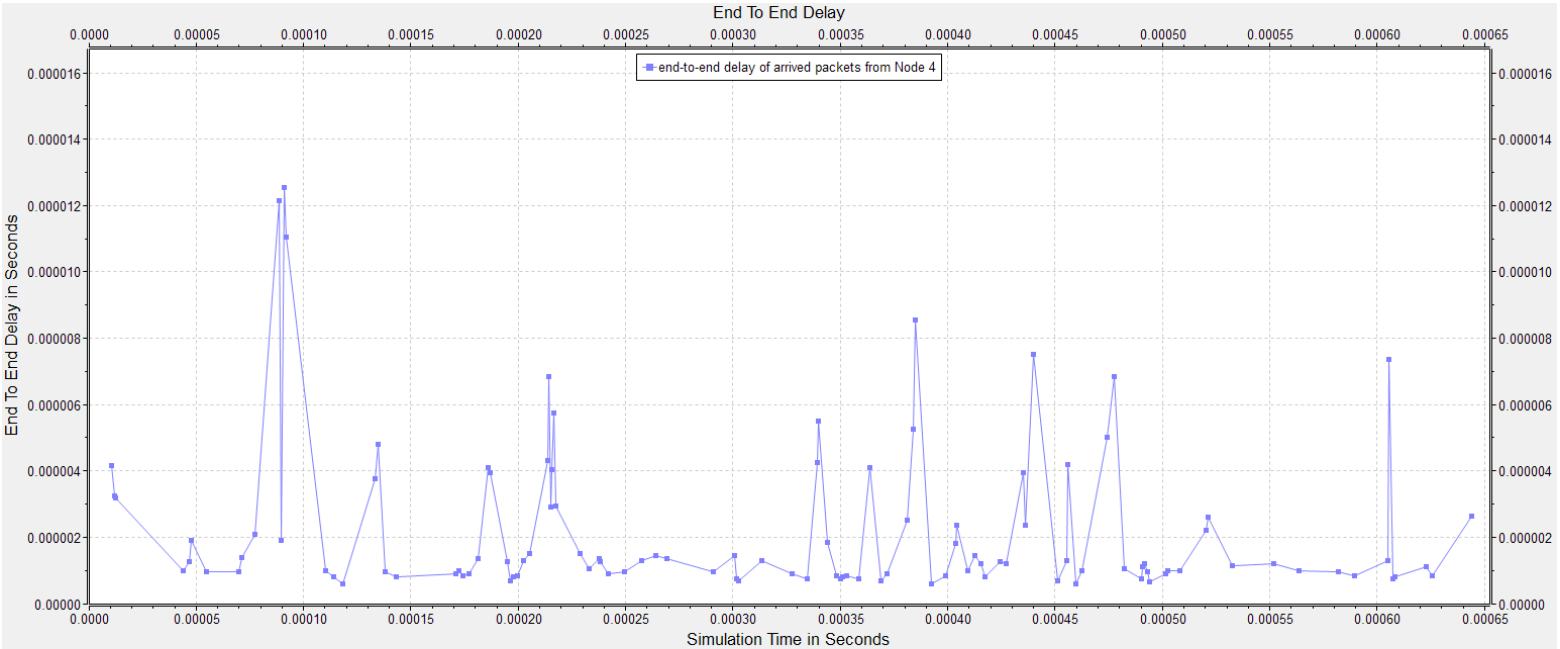


Figure A.11: Random Topology Static Routing End To End Delay of Packets from Node 4

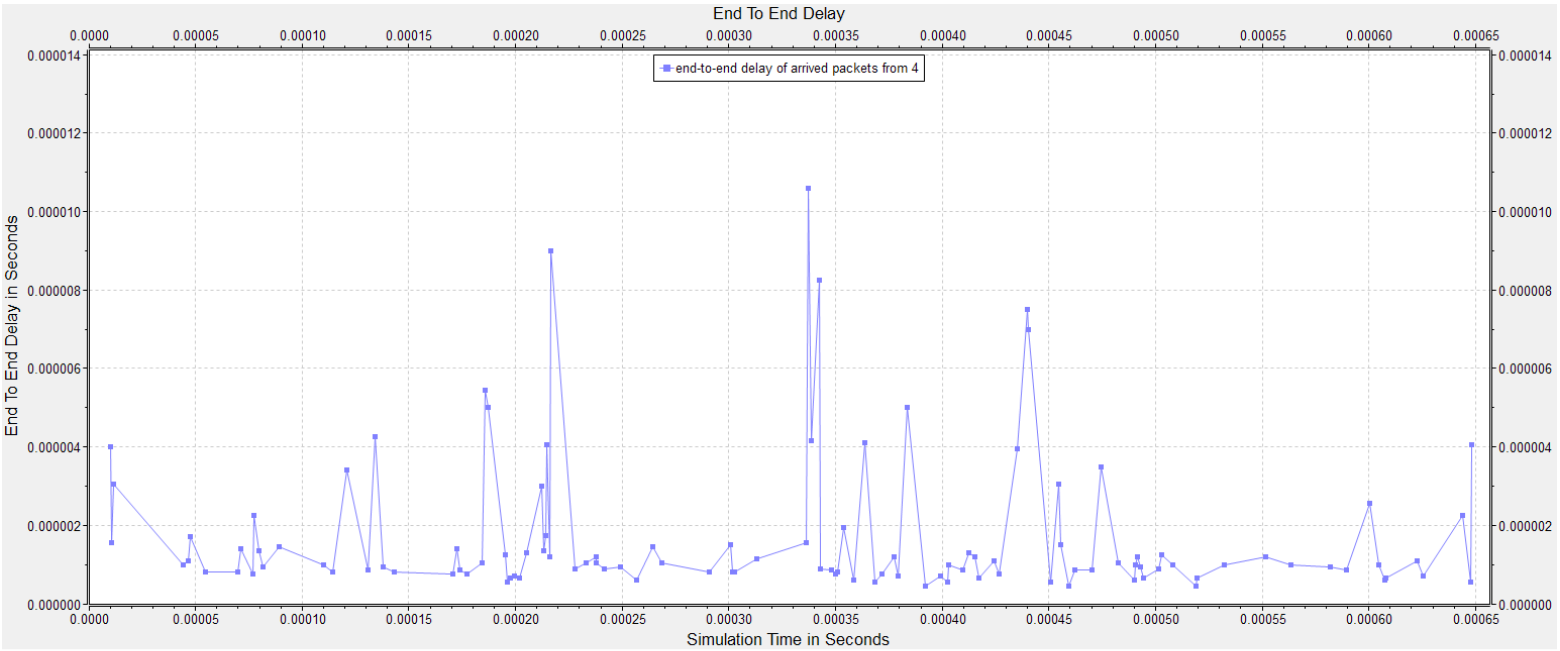


Figure A.12: Random Topology Dynamic Routing End To End Delay of Packets from Node 4

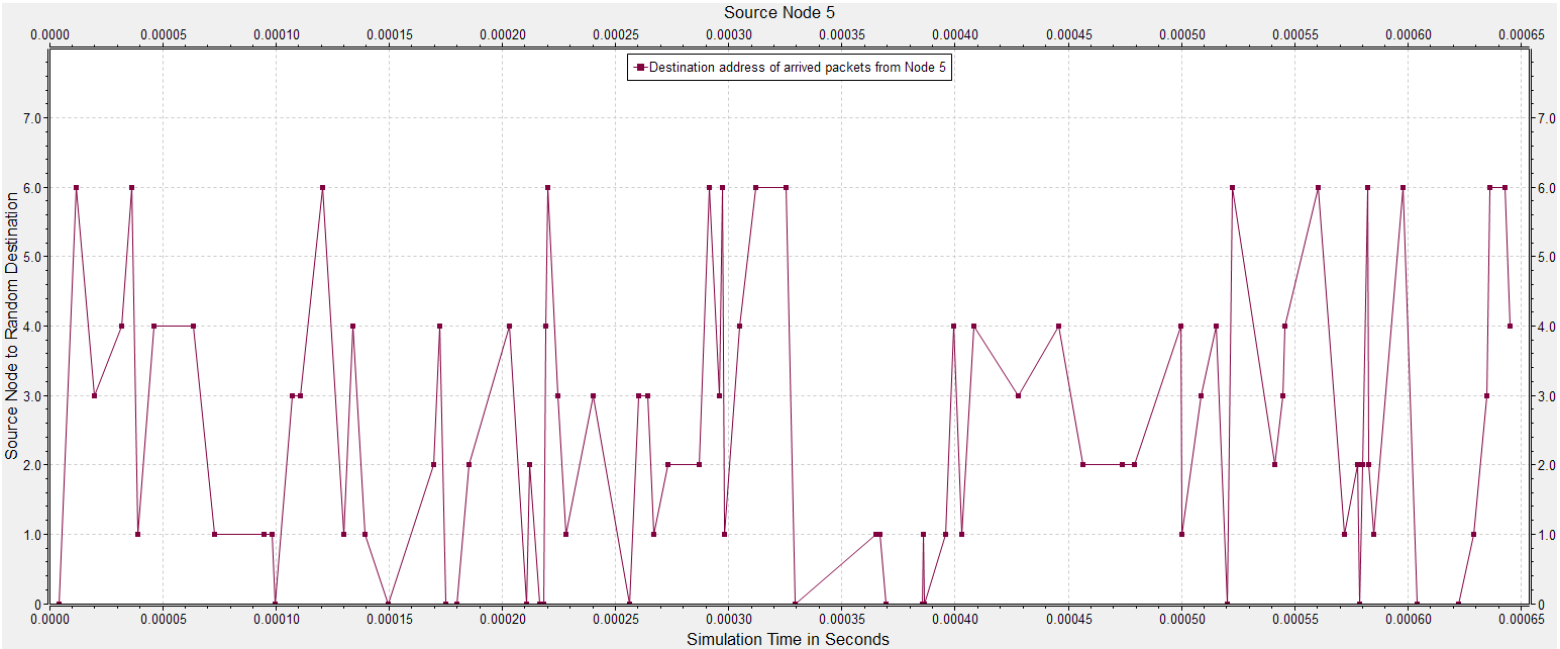


Figure A.13: Random Topology Data Sent from Node 5 to Random Destination

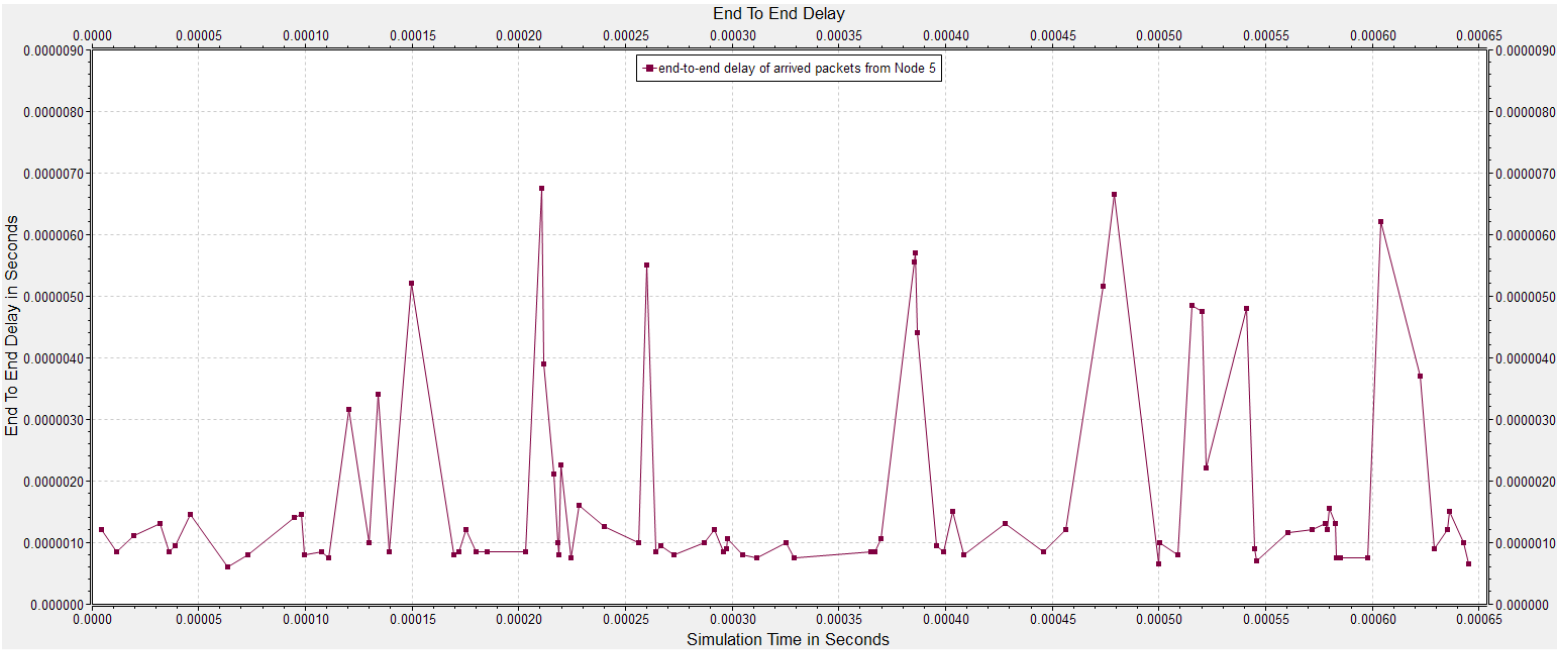


Figure A.14: Random Topology Static Routing End To End Delay of Packets from Node 5

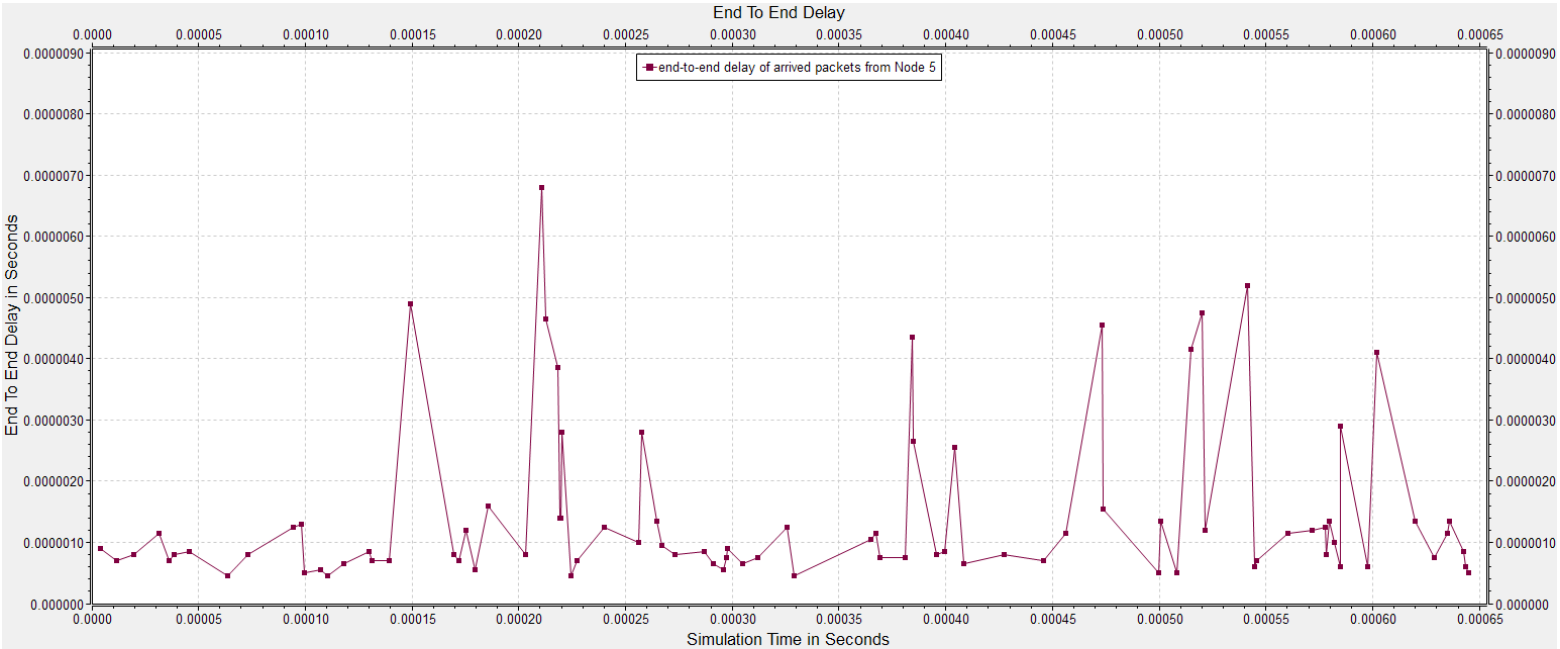


Figure A.15: Random Topology Dynamic Routing End To End Delay of Packets from Node 5

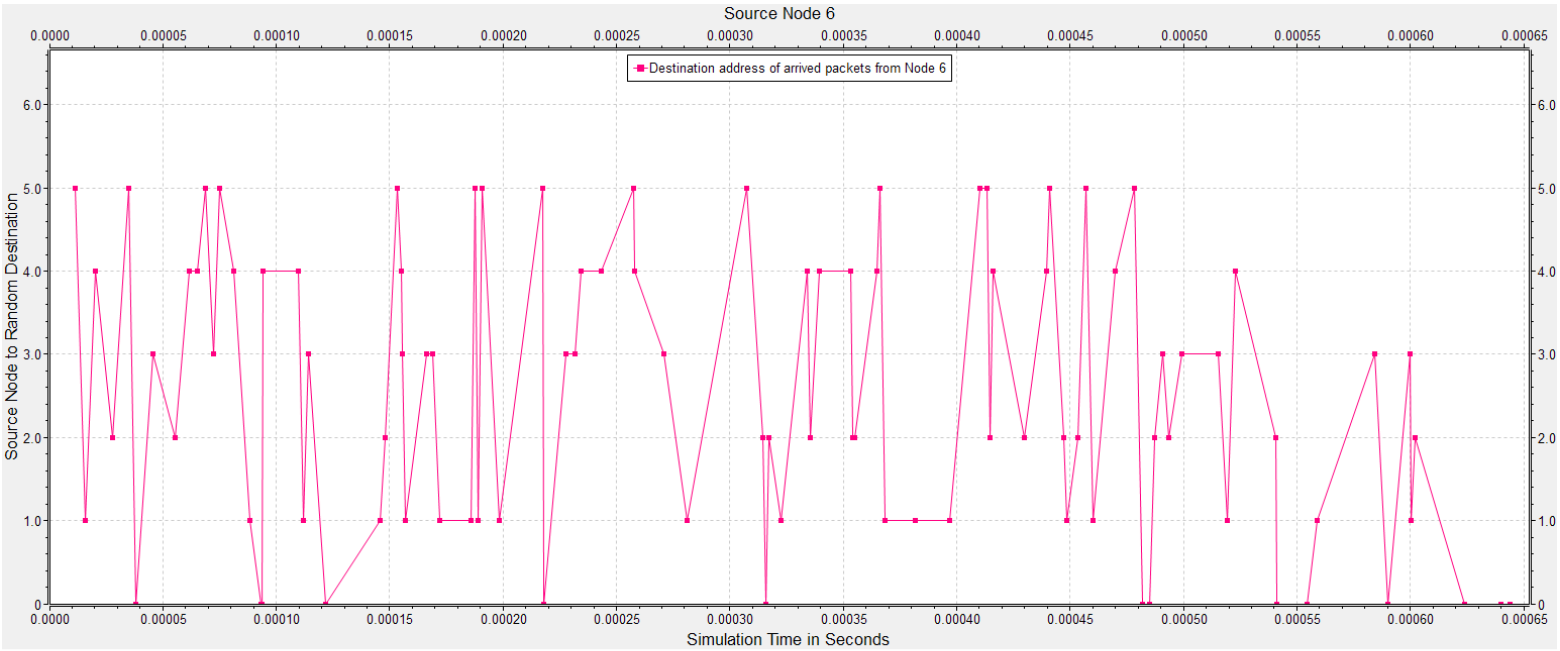


Figure A.16: Random Topology Data Sent from Node 6 to Random Destination

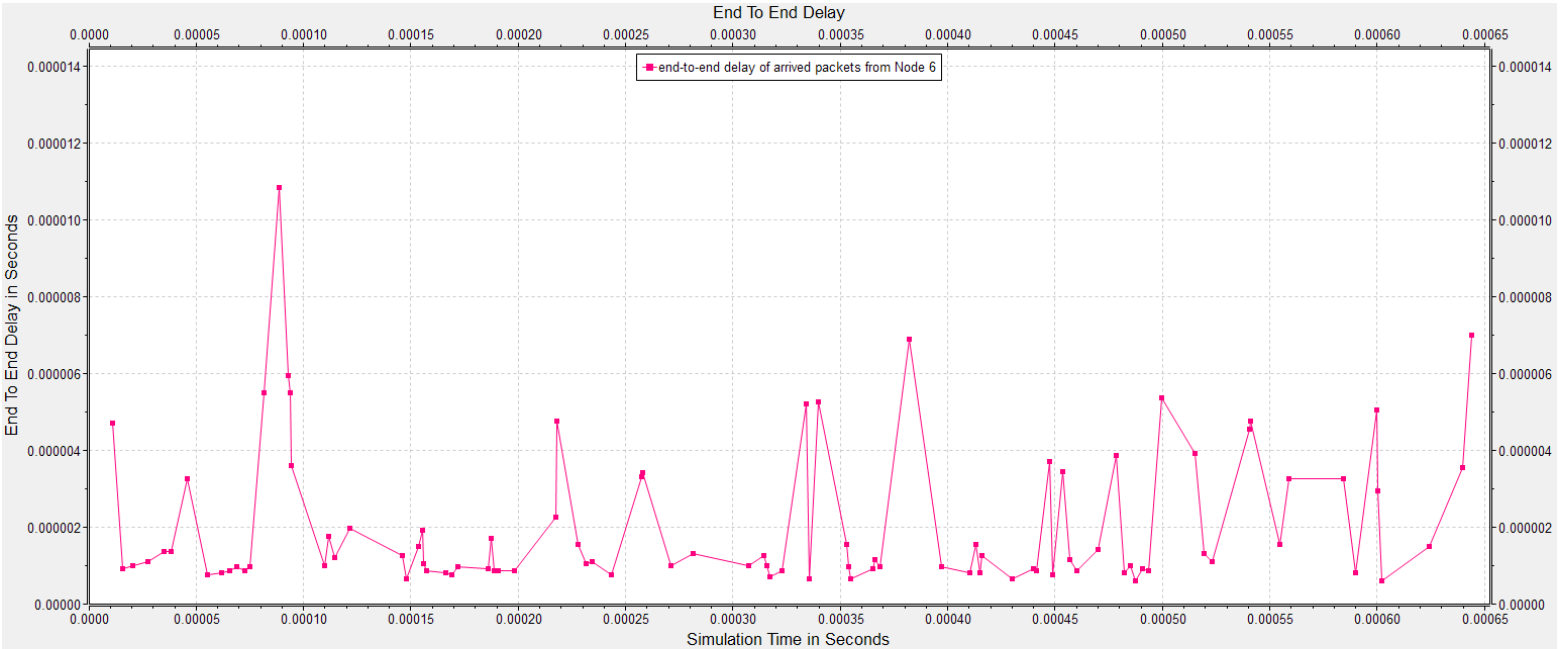


Figure A.17: Random Topology Static Routing End To End Delay of Packets from Node 6

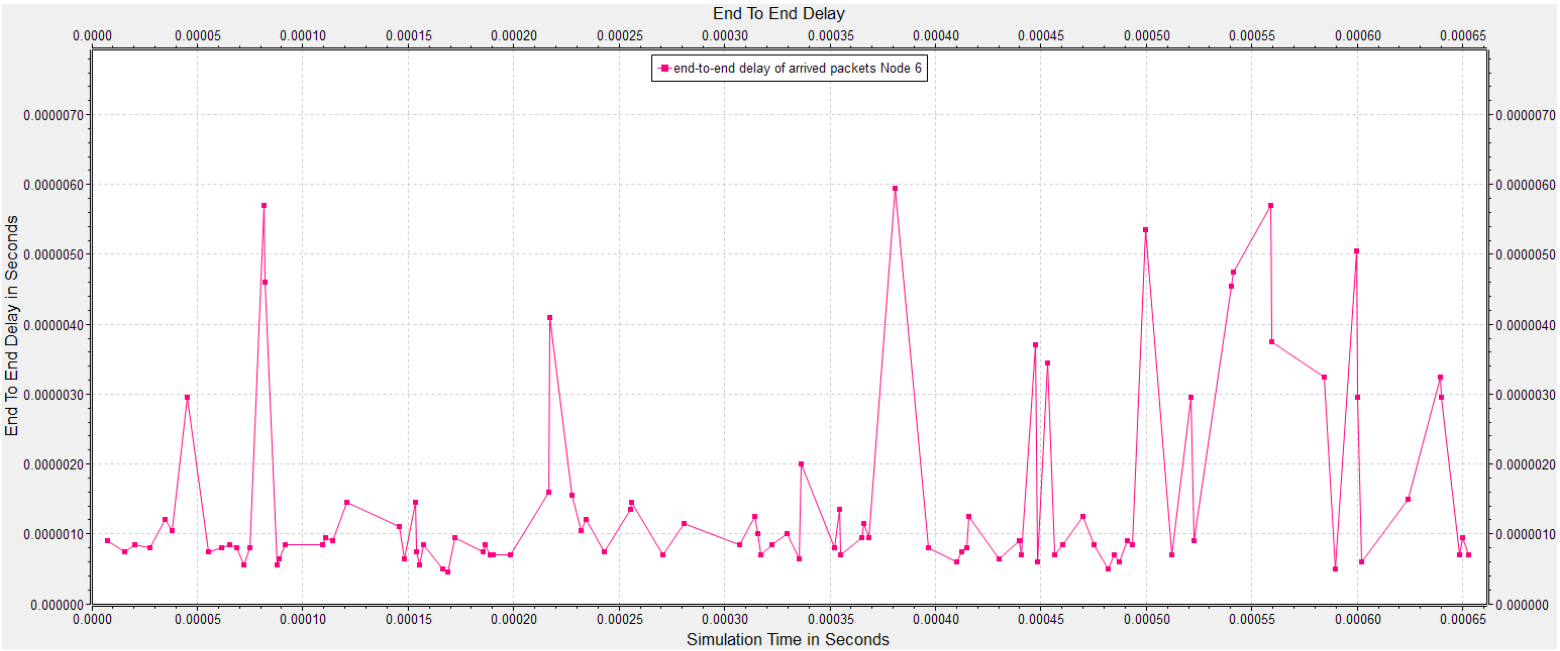


Figure A.18: Random Topology Dynamic Routing End To End Delay of Packets from Node 6

Appendix B

SpaceWire Brick Test

SpaceWire Brick is a USB to SpaceWire interface which is used for developement and testing purposes and helps visualize SpaceWire packets sent to and from the host computer.

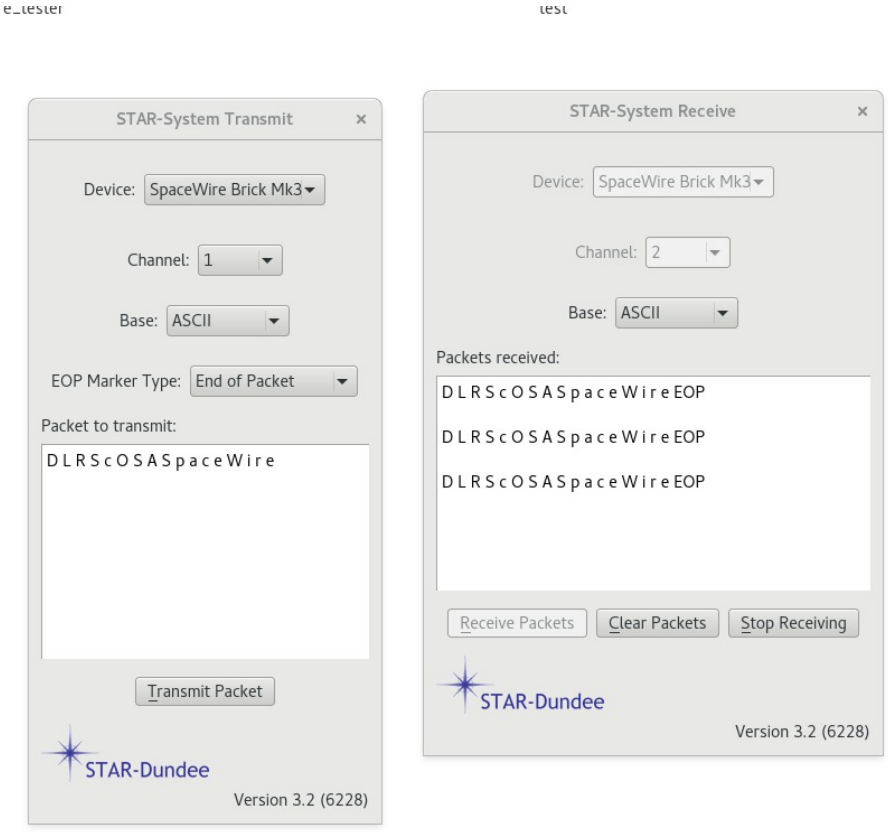


Figure B.1: SpaceWire Brick Test