A NEW FAULT-TOLERANT REAL-TIME ETHERNET PROTOCOL: DESIGN AND EVALUATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

EMRE ATİK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2021

Approval of the thesis:

A NEW FAULT-TOLERANT REAL-TIME ETHERNET PROTOCOL: DESIGN AND EVALUATION

submitted by **EMRE ATİK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Şenan Ece Güran Schmidt
Supervisor, **Electrical and Electronics Engineering, METU** _____

Prof. Dr. Klaus Verner Schmidt
Co-supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering, METU _____

Prof. Dr. Şenan Ece Güran Schmidt
Electrical and Electronics Engineering, METU _____

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Engineering, METU _____

Prof. Dr. Cüneyt Bazlamaçcı
Computer Engineering, İzmir Institute of Technology _____

Dr. Sibel Tarıyan Özyer
Computer Engineering, Cankaya University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Emre Atik

Signature :

ABSTRACT

A NEW FAULT-TOLERANT REAL-TIME ETHERNET PROTOCOL: DESIGN AND EVALUATION

Atik, Emre
M.S., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Şenan Ece Güran Schmidt
Co-Supervisor: Prof. Dr. Klaus Verner Schmidt

January 2021, 89 pages

This thesis is motivated by the communication requirements of contemporary real-time and embedded applications. These requirements are high-bandwidth, fault tolerance, determinism that allows schedulability and bounded latency, broadcast capability, accommodating sporadic traffic efficiently together with periodic traffic, and finally low cost, COTS interface hardware.

To this end, we propose a novel protocol, Shared Queue based Dynamic Slot Reservation (SQDSR) complete with message format, medium access and fault tolerance mechanisms. SQDSR implements a time-slotted synchronized communication layer over shared medium 100 Mbps Ethernet. SQDSR exploits the shared medium for broadcast capability, distributed scheduling of messages and fault tolerance. Furthermore, no switches are required that reduce the cost of implementation.

The performance of SQDSR is evaluated with OMNeT++ simulator with a realistic message set and node topology from an avionics application. To this end, we compare the performance of SQDSR to AFDX (Avionics Full-Duplex Switched Ether-

net) which is a widely used Ethernet-based communication protocol. The evaluation results show that SQDSR fulfills the requirements of the contemporary real-time embedded applications.

Keywords: Shared medium access, fault tolerance, dynamic slot reservation, real time communication, industrial Ethernet

ÖZ

YENİ BİR HATAYA DAYANIKLI GERÇEK ZAMANLI ETHERNET PROTOKOLÜ: TASARIM VE DEĞERLENDİRME

Atik, Emre
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Şenan Ece Güran Schmidt
Ortak Tez Yöneticisi: Prof. Dr. Klaus Verner Schmidt

Ocak 2021 , 89 sayfa

Modern gerçek zamanlı gömülü sistemlerin gereksinimleri bu tezin temel motivasyon kaynağıdır. Bu gereksinimler yüksek bant genişliği, hataya toleransı, kararlılığı, anons yayın yeteneği, hem düzenli hem düzensiz trafik tipleriyle uyumluluğu ve ticari olarak kullanıma hazır olmasıdır.

Bu çalışmada, Paylaşımlı Dizi Tabanlı Dinamik Ağ Dilimi Ayırma (SQDSR) yeni bir protokol olarak mesaj formatı, veri yolu erişimi ve hataya tolerans özellikleri ile birlikte önerilir. Ethernet tabanlı 100 Mbps bant genişliği ile zaman dilimli ve eş zamanlı bir haberleşme katmanı oluşturulur. Protokol veri yolunu anons yayın yapmak, mesajları düzenlemek ve hataları önlemek için kullanır. Ağ anahtarı gereksinimi olmaması düşük maliyet sağlar.

Önerilen protokolün performansı endüstriden alınan veriler kullanılarak OMNeT++ simülasyon modelleri ile değerlendirilir. Sonuçlar Ethernet tabanlı sıklıkla kullanılan bir protokol olan AFDX ile karşılaştırılır. Önerilen protokolün modern gerçek zamanlı gömülü sistemlerin gereksinimleri karşıladığı görülür.

Anahtar Kelimeler: Paylaşımlı ortam erişimi, hataya tolerans, dinamik ağ dilimi ayırma, gerçek zamanlı iletişim, endüstriyel Ethernet

To my family

ACKNOWLEDGMENTS

I would like to express my greatest gratitude to my supervisor Prof. Dr. Ece Güran Schmidt and co-supervisor Prof. Dr. Klaus Verner Schmidt. Without their outstanding guidance and support, this work would not have been succeeded.

I would like to thank ASELSAN Inc. for supporting my education. I appreciate the encouragement policy of the company towards post graduate research.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

AFDX	Avionics Full-Duplex Switched Ethernet
BAG	Bandwidth Allocation Gap
CSMA/CD	Carrier Sense Multiple Access Collision Detection
EOF	End of Frame
ES	End System
FPGA	Field Programmable Gate Array
FT	Fault Tolerance
LLC	Logical Link Control
OSI	Open Systems Interconnection
PES	Packetized Elementary Stream
PTP	Precision Time Protocol
SNAP	Subnetwork Access Protocol
SPoF	Single Point of Failure
SQDSR	Shared Queue based Dynamic Slot Reservation
SYNC	Synchronization
TDMA	Time Division Multiple Access
TSN	Time Sensitive Networking
VL	Virtual Link

CHAPTER 1

INTRODUCTION

Real-time and embedded applications are implemented on distributed components which exchange information to achieve the desired system behaviour. This message exchange is realized with a *communication network*.

The amount of data for such applications increase with the developments of new devices and sensors that can collect more and different types of information. Furthermore, video and images are also incorporated. Hence, a *high amount of network bandwidth* is required. These applications are often designed for life-critical dynamic systems where the communication network is required to provide *fault tolerance* and *determinism*. The high bandwidth together with determinism can together achieve the next requirement which is *low and bounded delay* for the messages. End-to-end message delays in the order of miliseconds are necessary to enable the correct operation of the control loops running on the distributed devices. The *time synchronization* of the nodes is an important support to satisfy such timing behaviour and delays. Furthermore, synchronization enables the coordination of the message transmission among nodes. The messages can be both periodic such as sensor data and sporadic such as messages produced by the actions of the users. The communication network should *efficiently support both periodic and sporadic message types*. *Broadcast/multicast* of these messages are required to coordinate the tasks that are running on the distributed nodes.

Finally, *COTS network interfaces* are preferred because of their compatibility, extensive testing record, availability and low cost. Furthermore network architectures that do not introduce devices such as switches in addition to the end-systems achieve *low cost and complexity*.

There is a large number of real-time embedded communication network standards that are developed both by the academic and industrial communities. Our literature survey in Chapter 2 shows that there is no single existing network architecture or standard that fulfill all of the requirements listed above.

The main contribution of this thesis is a novel real time protocol that we call Shared Queue based Dynamic Slot Reservation (SQDSR) that aims for addressing these requirements. SQDSR is implemented as a layer over full-duplex shared-medium 100 Mbps Ethernet. Hence, it features a bandwidth of 100 MBps, is inherently broadast capable and can be implemented by low cost COTS network interfaces. SQDSR features time slots for control and data communication where the lengths of the time slots carrying data frames are decided in run time to achieve efficient use of the network bandwidth. IEEE 1588 synchronization protocol is used for maintaining the time slotted structure. SQDSR introduces a novel slot reservation method for the sporadic messages by the means of a queue data structure that is consistently stored in all of the nodes. The queue contains the records of message send notifications of the nodes. These notifications are broadcast by the nodes and stored in the order of the message deadlines in the queue data structure by all of the nodes. The inherent broadcast capability of the shared medium is further exploited to support fault tolerance. If there are missing or lost messages, retransmission of those messages is scheduled by broadcast control messages allocating bandwidth on demand for efficient operation.

Scheduling different traffic types is not straightforward for bus based communication architectures. There are two prevalent real time scheduler types; cyclic and priority based [1]. Cyclic schedulers are used in time triggered networks that repeat the previously determined pattern to provide medium access to the communication nodes. On the other hand, the priority based schedulers are used in event triggered networks that dynamically apply the arbitration process to gain medium access. Though both have advantages and disadvantages, time triggered networks are known as more deterministic and preferred for systems with real time requirements. The reason behind it is the availability of assigned time slot for each network components in a predefined interval, which makes the worst case response time computable and the system deterministic. A deterministic system is the one whose time evolution can be predicted exactly [2], therefore, the transmission of a message is guaranteed to be realized in

a certain time interval. However, time triggered networks are generally less efficient solutions due to unused but allocated time slots. Plus, it is more difficult to update the network, i.e. adding or removing components, since it is non-trivial to update the schedule.

Besides from the type of scheduler for medium access control, the medium itself also plays an important role while designing a real time communication network. Today, the industry tends to prefer Ethernet based communication architectures rather than the bus based ones. The reason behind is its low cost, high bandwidth capability and strong compatibility. Although the lack of reliability for the original version of Ethernet CSMA/CD, many proposed alternatives exist to ensure reliable communication by incorporating either time division multiple access, token passing or master-slave architecture [3]. Yet, they all have several disadvantageous, which requires to seek new solutions.

The proposed protocol is intended for supporting both periodic and sporadic traffic by means of statically and dynamically allocated slots, respectively. The static and dynamic slots are repeated successively over each cycle with configurable length. The static part is scheduled in advance whereas dynamic slots are reserved based on the requests during run time. These requests could be notified either by predefined control slots or piggybacking to the end of any transmitted frame. The broadcasted nature and shared medium allows the notifications to reach all the components. Furthermore, it enables to intercept and report any failure thanks to constant monitoring of the network. The functionality of SQDSR depends on accurate time synchronization and reliable error management. Even though the protocol is not constructed on master-slave architecture, the time synchronization and error management operations should be handled by corresponding master nodes, which are configurable and alterable during run time.

In the scope of the thesis, the prevalent architectures and their pros and cons are detailed for both bus-based and Ethernet-based solutions. The protocols are evaluated in terms of reliability, efficiency and determinism. The background research paves the way for designing the new protocol. Therefore, it does not cover only introductory information on them, but there are also simulations and analyses performed. Next, the

proposed protocol SQDSR is introduced with detailed specifications. The protocol state machine is demonstrated with flowcharts and figures. Time synchronization and fault handling mechanisms are discussed. Finally, the evaluation part contains modelling and simulation of SQDSR using OMNeT++ [4]; an official network simulator and INET [5]; an open source library of network protocols and modules for OMNeT++. Special network traffic is obtained to perform real case simulation, supplied by a commercial company in the avionics industry. Avionics Full-Duplex Switched Ethernet (AFDX) is chosen as the benchmarking protocol since it is one of the most prevailing Ethernet based architectures deployed in real time systems, patented by Airbus [6]. The work concludes with the results of the comparison, the improvements offered by SQDSR and the possible future work. Briefly, the contribution of the thesis could be summarized as follows:

- Proposing a novel shared medium Ethernet based real-time communication protocol SQDSR for embedded systems complete with its message format and medium access arbitration mechanisms
- A novel decentralized time slot allocation method and enabling distributed fault management solution by means of broadcast communication
- An extensive comparative simulation study for the SQDSR under realistic message set and node functionality using OMNeT++ network simulator

The simulation results show that SQDSR fulfills the requirements of the real-time embedded applications.

CHAPTER 2

BACKGROUND

2.1 Real Time Safety Critical Systems

There is a growing trend in the embedded industry towards developing distributed systems with strict real time capabilities. Distributing system components does not only increase the modularity and maintainability, but it is also compulsory to prevent single point of failure (SPoF). Such systems require network architectures to enable the information exchange among the system components. Deterministic real time communication and fault tolerance for reliability are the primary requirements. A system meets real time requirements if both execution of tasks and transmission of messages have deterministic and bounded worst case response times [1]. The consequences of missing the deadline determine the degree of real time capability; soft real time and hard real time. For the latter, following the real time constraints is a must for the system validity and missing a deadline is unacceptable.

Ensuring safety for the distributed embedded systems is another critical concern. It is expected that the system continuously operates without any fault or handles faults appropriately and returns to its normal operation. The mechanisms and techniques that enable the system deliver consistent service even in the presence of fault is called Fault Tolerance (FT) [1]. FT is achieved using different types of redundancy; hardware, software, time and information. Hardware redundancy is provided by replication of components such as nodes, switches, communication links. Software redundancy contains multiple tasks to do same operations to have a backup state to restore from fault. Time redundancy performs actions several times with the same hardware and software, such as retransmission of a message. Finally, information redundancy

is provided by means of storing additional information to restore from fault state [1].

The replication of hardware or software components is categorized under active and passive replication. If the same redundant operation is carried out in parallel, it is referred to as active replication. Hence, the failover time, i.e. the act of switching to a redundant system, is zero for active replication. The main disadvantage of this method is that the selection algorithm between one of the redundant operations needs to be defined and applied in normal operating conditions. On the other hand, the passive replication is activated in case any fault occurs. Therefore, there is a non zero failover time, which is the main disadvantage of passive replication.

2.2 Bus Based Communication Architectures

There are several different communication architectures to provide special requirements. They could differ by the implementation of physical layer, media access control, error detection and correction. It is expected that they satisfy the properties of reliability, flexibility and scalability. In the sense of these concerns, we focus on widely used bus based communication architectures including Mil-Std-1553, SAFEBus, TTP/C, FlexRay, TTCAN, FireWire and SpaceWire. After giving a brief information on each, they will be analyzed by disclosing their advantages and disadvantages based on the requirements. Furthermore, several simulations and analyses are performed for Mil-Std-1553 since it influences the dynamic slot reservation feature of the proposed protocol SQDSR as introduced in Chapter 3.

2.2.1 Mil-Std-1553

Mil-Std-1553 represents one of the first communication data bus standards which is extremely reliable and widely used in military and space applications including Space Shuttle and the International Space Station [2]. The communication over bus is half duplex master slave and limited to 1 Mbps, but there are improvements reported to reach up to 100 Mbps with transceiver upgrade with new CMOS technology [7]. There is a single bus controller supervising the bus access to the nodes. The protocol contains a query cycle to ask each node whether they have any packet to transmit or

not. Each node that returns true for the query part is asked to transmit one by one. Each query consists of 2 control packets, while each transmission consists of 4 control packets plus data packet. Mil-Std-1553 packets are named as word and a word has 20 bits length with 4 bits header and 16 bits payload. Since data packets consist of words, the minimum data length is 2 bytes while maximum data length for each turn is restricted with 64 bytes by the protocol. Mil-Std-1553 operates with at most 32 nodes including the master. One of the nodes could be assigned as a backup master, which makes the master assignment possible during run time [8].

Mil-Std-1553 is considered as a reliable protocol due to the master slave polling request response structure. Since the nodes are only allowed to respond after the command, there is no possibility of collision. Furthermore, the protocol also supports dual redundancy. In this configuration, the redundant channel is not always active but only in case of failure. The secondary bus could be used to remove or tolerate the fault in case of babbling or failure on the primary bus. Any message failure could be detected by status bit, but the protocol does not tolerate message loss and retransmission is required [2]. As a prevalent safety protocol, Mil-Std-1553 has many inspirational characteristics and techniques for the design of a fault tolerant real time communication architecture, therefore, it needs to be examined with details. Next, the network capacity of Mil-Std-1553 and performed simulation results will be discussed in the scope of the thesis.

In this part of the thesis, we try to model Mil-Std-1553 in deterministic network conditions. The polling request response cycle and transmission cycle together construct a repeating periodic cycle (T) and it's strongly related with number of nodes who have data to transmit. Hence, the periodic cycle is referred to as T(n) where n is the number of nodes that return true to in the query cycle. The period is related to the total number of nodes (N), control packet length (C_p), data packet length (D_p) and bus bit rate (B) as follows:

$$T(\text{n}) = \frac{N \cdot (2 \cdot Cp) + n \cdot (4 \cdot Cp + Dp)}{B} \tag{2.2.1}$$

Assume that all the nodes generate packets uniformly. For the maximum possible arrival rate, each node always has some packet to transmit in each cycle. Then, the

equation is updated as :

$$T(\mathrm{N}) = \frac{N \cdot (2 \cdot Cp) + N \cdot (4 \cdot Cp + Dp)}{B} \tag{2.2.2}$$

Each node could only transmit one packet per T. Hence, for a stable network, it is only allowed to generate maximum N packets in each interval T; i.e. maximum arrival rate (λ_{max}) :

$$\lambda_{max} = \frac{N}{T} \tag{2.2.3}$$

$$\lambda_{max} = \frac{N \cdot B}{N \cdot (2 \cdot Cp) + N \cdot (4 \cdot Cp + Dp)} \tag{2.2.4}$$

$$\lambda_{max} = \frac{B}{6 \cdot Cp + Dp} \tag{2.2.5}$$

Mil-Std-1553 serves as a reference point for the proposed protocol in this thesis. To this end, we perform a simulation study of Mil-Std-1553 to demonstrate its capabilities. In this thesis, we use OMNeT++ [4] for performance evaluation as we introduce in detail in Chapter 4. To the best of our knowledge, there is no published and verified Mil-Std-1553 model for OMNeT++. Therefore, we developed a simulator that we call *NetworkSimulator* in the scope of this thesis to perform analysis for Mil-Std-1553 protocol.

NetworkSimulator is designed to be generic with respect to the media access protocol. The protocol and parameters should be easily updated via a configuration file. The network packets are produced based on packet arrival rate and packet arrival distribution where uniform and exponential distribution are supported. *NetworkSimulator* has a configurable clock cycle that determines the granularity of the system. For each cycle, several packets can be generated based on the packet arrival rate. The packet properties as source node, destination node, deadline, priority, data length are assigned randomly among the constraints indicated in the configuration file. Based on the media access protocol used, each node tries to transmit the packets on their

queues. Moreover, *NetworkSimulator* has a statistics feature to calculate performance metrics of the designed system. The results contains average latency, maximum latency, average queuing delay, average jitter, real data bit rate, dropped and expired packets count.

The command response protocol is implemented as a subclass of the abstract media access protocol class of *NetworkSimulator* in order to simulate Mil-Std-1553. The aim of the simulator is finding the maximum possible arrival rate the network supports to meet the latency requirements by considering various data lengths. The deadline of messages is set to the 50 μs to observe the number of expired packets detected by *NetworkSimulator*. The arrival rates are selected within the constraint of the stable network conditions as (2.2.5). Clock cycle is set to 0.01 μs and physical layer bit rate capacity to 100 Mbit/s. Since Mil-Std-1553 supports a maximum of 32 nodes, the number of selected nodes is 32 to find the worst case results. The expired packet percentage is measured in the case of maximum length data packets, minimum length data packets and uniformly distributed length data packets as shown in figures 2.1, 2.2, 2.3.

For the maximum data length (64 bytes), the simulation results show that an arrival rate of more than 32 packet/ms results in more than 1 percent expired packets. For the minimum data length (2 bytes), the arrival rate is increased to 270 packet/ms for the same performance. If the packet lengths are uniformly distributed between 2 to 64 bytes, than the results shows 80 packet/ms causes more than 1 percent expired packets. The results are obtained with 5000 packets per simulation providing a sample mean within 2% of the true mean with probability of 99% based on the confidence calculation explained in Section (4.1.3).

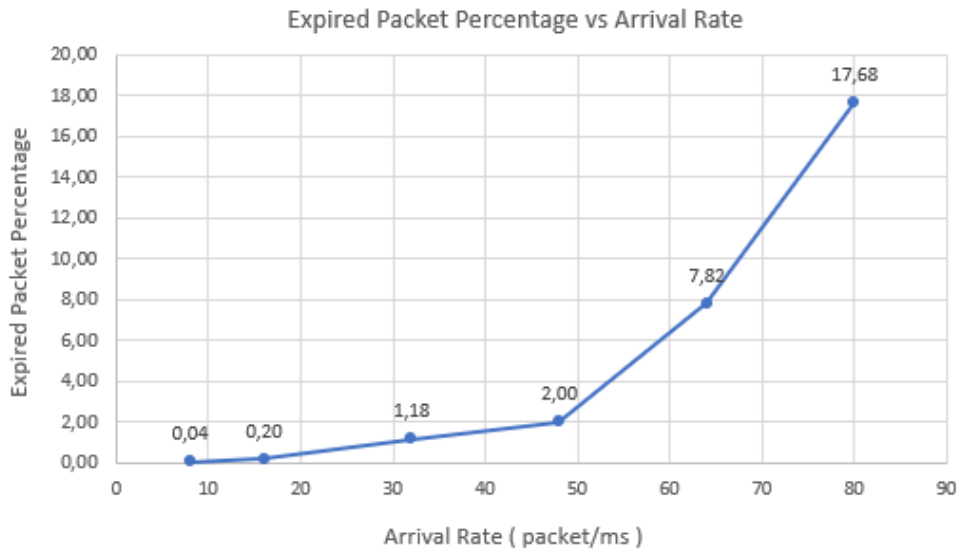

Figure 2.1: Mil-Std-1553 Expired Packet Percentage with Maximum Packet Length

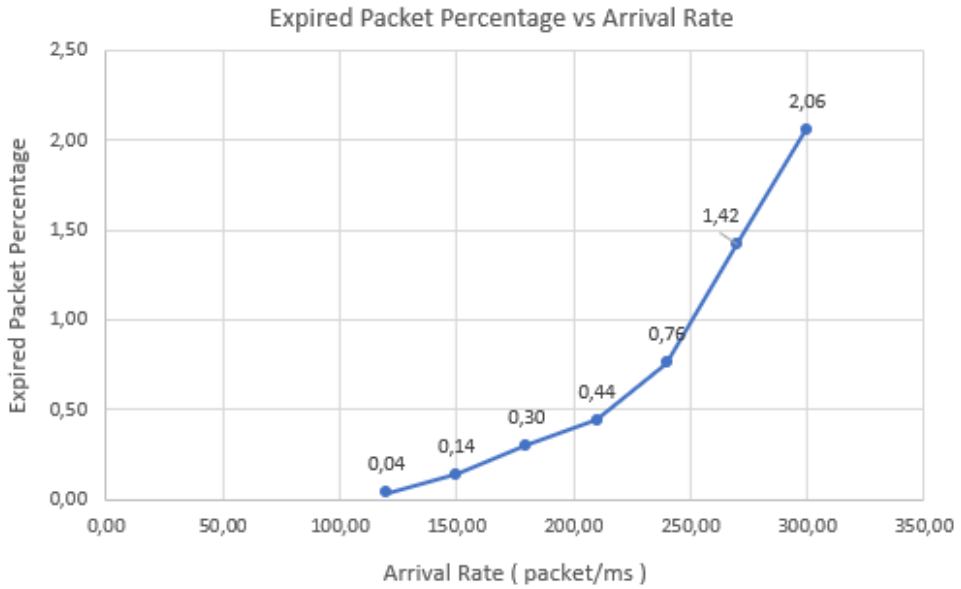

Figure 2.2: Mil-Std-1553 Expired Packet Percentage with Minimum Packet Length

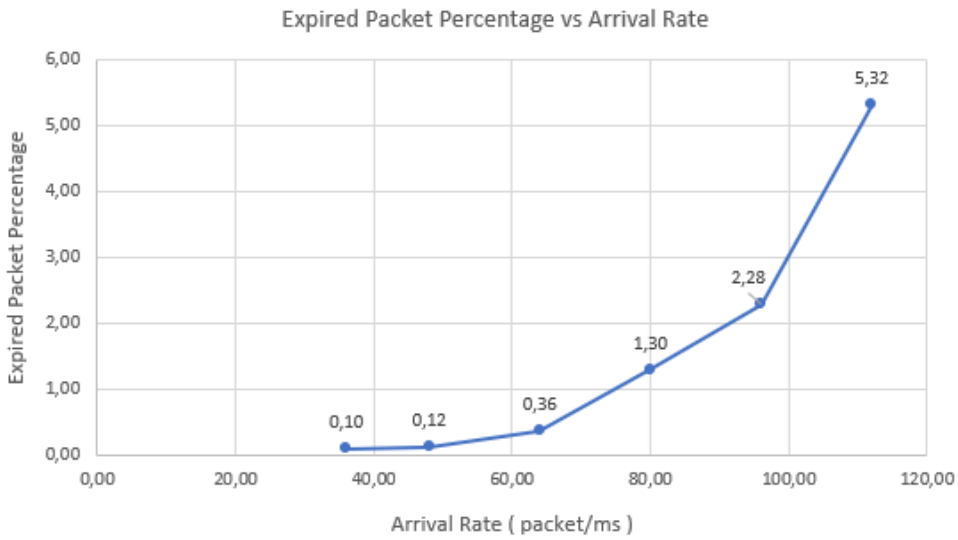

Figure 2.3: Mil-Std-1553 Expired Packet Percentage with Uniformly Distributed Packet Length

The simulation puts forward essential results for the performance of Mil-Std-1553 in high loaded stable network conditions. The limited arrival rates are found in return for compromising packet lost. Mil-Std-1553 has high efficiency if each node has a packet to transmit. Plus, Mil-Std-1553 has only 20 bits length control packets which gives a significant advantage over the Ethernet based protocols. However, the command response cycle is a useless interval, if the queues of the nodes are empty. Plus, the command response cycle is not the only cycle that uses the control packets, but also the data transmission cycle requires commands between master and source node, master and destination node. An other disadvantage of the protocol is its dependency on special hardware devices, therefore, incompatibility with the other protocols. Hence, Mil-Std-1553 is not an ideal protocol with regard to efficiency and compatibility although it has a good reputation of being reliable and redundant.

2.2.2 SAFEBus

SAFEBus [2], is a backplane bus in a computing cluster, registered trademark of Honeywell Corporation, used in safety critical functions on commercial aircrafts. It has masterless, quad redundant bus with 60 MB/s limited data rate. Full duplication

of bus interface units (BUIs) in each node exists that act as a bus guardian for the other BIU to prevent any babbling or transmitting erroneous data. BIU also check the data transmitted and received for errors. They also control their partner's access to bus lines. The system is designed as fault tolerant such that its standard is guaranteed to tolerate one failure and may tolerate multiple faults. If any message is lost, it is handled by quad redundant bus, therefore, retransmission is not required. If any node breaks down, the failing node will be fail-silent and removes itself from operation. The other nodes detect the situation by loss in scheduled communication and will continue their normal operation without any problem.

One of the apparent disadvantages of SAFEBus is its expensive architecture and proprietary components. Plus, it has a limited network length less than 1.5 meter. SAFEBus is not an extensible protocol during run time since adding a new node requires a new design of schedule. Nevertheless, it is possible to replace any failed node with the same type of node. Hence, SAFEBus does not meet the conditions if the system requires off-the-shelf components, extensibility and satisfying network length.

2.2.3 TTP/C

Time Triggered Protocol with Society of Automotive Engineers Class C Requirements (TTP/C) [2], is a time triggered, masterless communication protocol that is mostly used in the automotive industry. It is designed independently from the physical layer. It has a high level of reliability and could tolerate multiple faults thanks to its dual redundancy. The supported bit rate is between 5-25 Mbps, but could enhance up to 1 Gbps with Gigabit Ethernet. It provides autonomous message transmission with known delay and bounded jitter.

The protocol could cope with node failure with bus guardians that also report the failure to the other nodes. The failed node will be fail-silent if any problem has occurred. Any message failure could also be detected by a status bit and global acknowledgement. As a time triggered protocol, it requires pre-design of schedule, therefore, adding a new node during run-time is not possible. Nevertheless, it is possible to replace any failed node with the same type of node.

Not supporting event triggered communication and relying on TDMA slotted architecture are the main disadvantages of TTP/C. Slots reserved for sporadic messages could be idle if the assigned nodes have nothing to transmit. Hence, TTP/C is not an efficient protocol in terms of utilization. Furthermore, in spite of commercially available components, it should be licensed to use.

2.2.4 FlexRay

FlexRay [2], is designed to be used in the automotive industry by the FlexRay consortium and only available to the member of consortium. The protocol supports both time triggered and event triggered communication as synchronous frames and asynchronous frames in a single communication cycle. Synchronous frames exist in the static segment that each slot is assigned to specific node at design time. The static segment is followed by the dynamic segment which consists of mini slots for event triggered frames based on arbitration. The protocol is tolerant to node failure; the communication proceeds between remaining nodes. It is also tolerant to message failure if the redundant channel exists.

Although the protocol is only allowed to be used in the consortium members, it has importance for leading the way of using synchronous and asynchronous communication together. Apart from unavailability, its supported limited network length and bit rate are the other cons of the protocol. Moreover, the arbitration process in the dynamic segment is priority based, i.e. comparing message IDs of packet. It could lead the messages with low priority miss the deadlines.

2.2.5 TTCAN

TTCAN [2], is an extension to the standard CAN with time triggered communication capability. TTCAN uses the modified CAN controllers enhanced with a frame synchronization entity. There are slots to transmit periodic messages, slots to transmit event triggered messages. Likewise CAN, arbitration based on message priority is applied on event triggered section. Supporting both event triggered and time triggered communication makes it resemble FlexRay. Plus, it is publicly available and

has off-the-self components.

The protocols having master should have a backup master, which will be activated if the master breaks down, for the reliability purposes. TTCAN has a time master to achieve clock synchronization which periodically sends a reference frame to begin the communication cycle. In case of failure, the nodes which are configured as redundant time masters try to be the master by sending the reference message; and the one that wins the arbitration will be the new master. Briefly, the protocol is worth mentioning as publicly available alternative to FlexRay. However, it has limited bit rate of maximum 1 Mbps and constrained network length. Plus, the standard does not include any redundancy management.

2.2.6 FireWire

FireWire (IEEE 1394) [2], is a communication architecture with fast communication rates up to 3.2 Gbps. It is commonly used in consumer electronics and avionics systems due to low cost high bandwidth capability. FireWire speeds up the arbitration process by using bidirectional communication in which arbitration frames are sent while data frames are being sent. It has isochronous transmission phase for broadcast high speed data transmission without checking errors and asynchronous transmission phase for error-free peer to peer communication.

The protocol is not able to operate in fault tolerant mode and has no fault hypothesis. However, node failure is detected by arbitration timeout or cable bias voltage drop. In asynchronous mode, message failure is detected by loss of acknowledgement message. The protocol does not provide redundancy but there could be an unused loop that activates if any failure occurred in tree topology. The protocol allows adding removing nodes during run time. When a node is removed or added, a bus reset process is automatically initiated, starting a new bus and node auto-identification procedure to provide newly added node's address to the other nodes, select root nodes and isochronous master node.

Using full duplex bidirectional communication to speed up arbitration process puts FireWire forward among other protocols. Even though it lacks of redundancy and

strict reliability, it could be preferred because of its high speed data transmission and support of both isochronous and asynchronous modes. Additionally, it requires license to be employed.

2.2.7 SpaceWire

SpaceWire [2], is an event triggered masterless data transmission architecture that operates with cascades of hubs and switches. In addition to switch based architecture, the protocol also provides peer to peer communication. It has found applications on aerospace industry including NASA's spacecrafts. SpaceWire supports high speed bit rate and regulates message flow with control tokens.

SpaceWire supports redundancy management and handles node failure. The failed node could be replaced with the same type of node but adding a new node is not possible since it requires update on routing switch configuration. The dependency to switches to operate introduces indeterminism to the protocol. Furthermore, SpaceWire is not compatible with the other communication architectures. It has a special frame format and requires special components to operate.

2.3 Ethernet Based Communication Architectures

Ethernet, or IEEE 802.3 is layer-2 media access protocol that offer high flexibility, compatibility and ability to connect many nodes at long distances. Ethernet promises higher bandwidth and performance and lower component costs compared to field-bus technologies that are currently replaced by Industrial Ethernet. However, standard Ethernet does not provide real-time characteristics due to the nature of shared medium access algorithm CSMA/CD. If more than one node start transmitting simultaneously, the collision is detected and causes the node to wait for a random amount of time based on exponential back off. Hence, it is not possible to know the worst case response time and whether or not the messages meet the deadlines. Modern Ethernet now employs full-duplex links and switches to overcome the effects of CSMA/CD. Although an improvement, the non-deterministic factor is still present due to the unknown latency in switch queues [9].

To bring the real time characteristics to Ethernet, there are three type of solutions applied to the standard Ethernet [10], as shown in Fig. 2.4 cited from [10]. The first option utilizes standard Ethernet and TCP/UDP/IP protocols. The real time mechanism is provided on application layer but it has limited performance. The second solution contains a special data protocol on top of Ethernet plus the real time services on application layer. The unchanged Ethernet layer provides adapting standard Ethernet without any modification but limited performance compared to the last solution. It is the one with modified Ethernet layer plus special top layers. Despite compatibility issues, this solution generally results in higher real time performance.

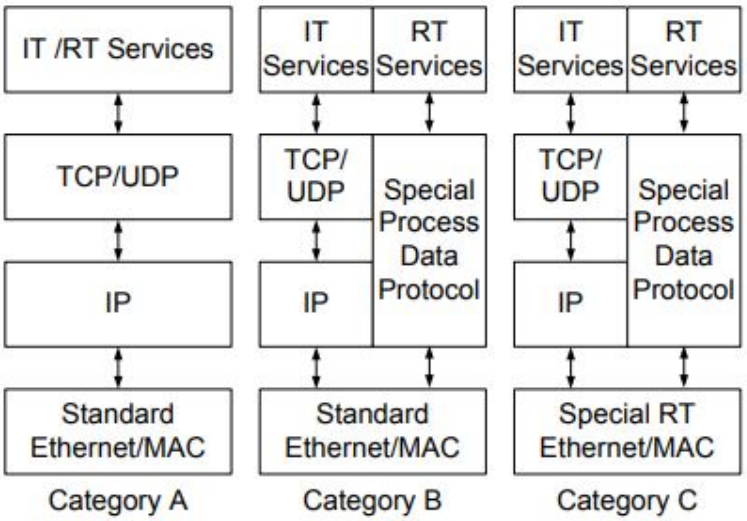

Figure 2.4: Categorization of Industrial Ethernet Systems in terms of Software and Hardware Needs

The industrial Ethernet protocols develop new access methods rather than non-deterministic CSMA/CD. Major industrial Ethernet protocols EtherNet/IP, Ethernet Powerlink, PROFINET RT/IRT, EtherCAT, SERCOS III, Modbus/TCP implement these methods by either adding a new layer on top of MAC or update the MAC itself. For the scope of the thesis, prevailing Ethernet based solutions; Ethernet Powerlink, EtherCAT, AFDX and TSN protocols will be discussed.

2.3.1 Ethernet Powerlink

Ethernet Powerlink [11], is a software based master-slave solution to provide real time characteristic to standard Ethernet, as category B in Fig 2.4. The master gives permission to slaves sequentially in the isochronous phase. This mechanism allows only one node to transmit in each time slot, therefore, there is no possibility of collision. There is also an asynchronous phase where standard Ethernet communication takes place.

Powerlink resembles Mil-Std-1553 including polling request-response between master-slave. The main difference is that the node responds directly with its data via Ethernet frame after the request. Still, it suffers from inefficiency of the command response cycle. To make polling request response structure more efficient, 'Poll Response Chaining' methodology is implemented such that all polling data of managing node is combined into a single broadcast frame, as shown in Fig. 2.5 from [11]. The slave nodes could send their responses at a specified point of time with fixed predetermined time delays.

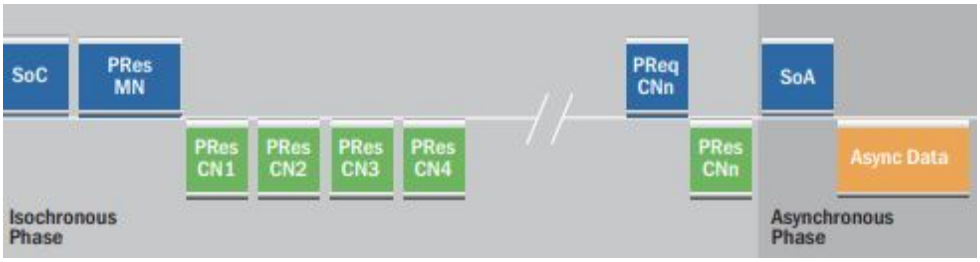

Figure 2.5: Ethernet Powerlink Polling Response Chain

The protocol supports both dual redundancy and master redundancy. The redundant master remains on hot standby and monitors the master's operation to take action if necessary. The master and slave nodes do not require special hardware to implement and the software implementations are publicly available [12]. Besides, Ethernet Powerlink is hot-plug capable; the nodes could be added or removed during run time [10].

2.3.2 EtherCAT

EtherCAT is a master slave based layer-2 protocol that requires implementation on dedicated hardware for slaves [9]. There is no command response cycle as in Powerlink or Mil-Std-1553, instead, the master is always the only node allowed to start a frame. This frame passes through all nodes in sequence where each node reads the data addressed to it and writes its data back to the frame all while the frame is moving downstream [13]. During each cycle, relevant output data is extracted by the devices from the Ethernet data packets sent by the bus master. Input data is also stuffed into packets "on the fly"; these packets arrive again at the bus master upon reaching the end of the ring. As this processing is done "on-the-fly", a specialized hardware is required. Fig. 2.6 from [13] shows EtherCAT frame with successive datagrams assigned to slaves. The datagrams contain control and data sections where they are also divided into send and receive parts. EtherCAT slaves write to send and read from receive parts of the datagram. Hence, single datagram consists of two control sections and two data sections.

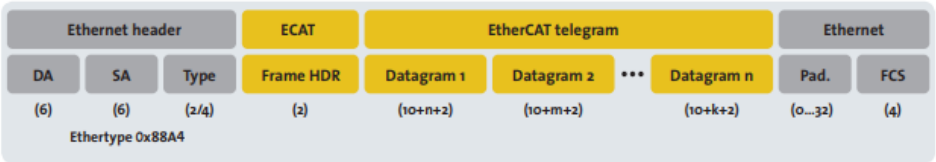

Figure 2.6: EtherCAT Ethernet Frame

EtherCAT is a summation frame protocol that does not support direct cross-traffic, i.e. direct communication between nodes without having to go through a master, which increases the overall data traffic on the network. If frame is corrupted, summation frame protocols always lose the entire cycle. EtherCAT is optimized for applications with only very low network traffic volume. In systems with a heavier data load, there is a disproportionate rise in cycle times. It suffers greatly from the lack of direct cross-traffic due to the duplicate data in send and receive parts of datagram, which sharply reduces the performance. To sum up, EtherCAT does not fit the required qualifications of efficiency and compatibility.

2.3.3 Time Sensitive Networking

Time Sensitive Networking is one of the emerging standards to overcome the indeterminism in Ethernet while using its power of interoperability. It is a pure layer-2 protocol that supports time-critical and non-critical traffic at the same time. TSN improves the standard Ethernet by means of IEEE 802.1 extensions given in Fig.2.7 [14]; including timing, synchronization, forwarding, queuing and redundancy. The standard provides guaranteed message transmission through switched Ethernet based network. TSN has an IP core with embedded software package providing TSN functionality to FPGA devices.

Standard	Area of Definition	Title of Standard
IEEE 802.1ASrev, IEEE 1588	Timing and synchronization	Enhancements and performance improvements
IEEE 802.1Qbu and IEEE 802.3br	Forwarding and queuing	Frame preemption
IEEE 802.1Qbv	Forwarding and queuing	Enhancements for scheduled traffic
IEEE 802.1Qca	Path control and reservation	Path control and reservation
IEEE 802.1Qcc	Central configuration method	Enhancements and performance improvements
IEEE 802.1Qci	Time-based ingress policing	Per-stream filtering and policing
IEEE 802.1CB	Seamless redundancy	Frame replication and elimination for reliability

Figure 2.7: Time Sensitive Networking IEEE 802.1 Standards

The extensions IEEE 802.1AS and IEEE 802.1Qbv are responsible for accurate time synchronization and traffic scheduling. The switches hold the messages in queues until the corresponding schedule time as represented in Fig. 2.8 [15]. IEEE 802.1Qbv provides guaranteed bound for message latency in switches, which the worst case latency for time critical data is 100 μs. IEEE 802.1Qbv provides best effort service to non-critical data. IEEE 802.1CB provides required redundancy by transmitting multiple copies of messages. The first message copy is processed and the others are discarded. The TSN 802.1Qbu includes preemption feature to increase the bandwidth of low priority messages against large high priority packets. Preemption makes the fragmentation of large packets possible to maximize the bandwidth for all type of net-

work packets. The extension IEEE 802.1Qca and 802.1Qci protect against failure in nodes or switches by isolating them from the system. IEEE 802.1Qch is responsible for traffic shaping. IEEE 802.1Qcc is the stream reservation protocol that identifies traffic class between specific source and destination. It focuses on network management and administration.

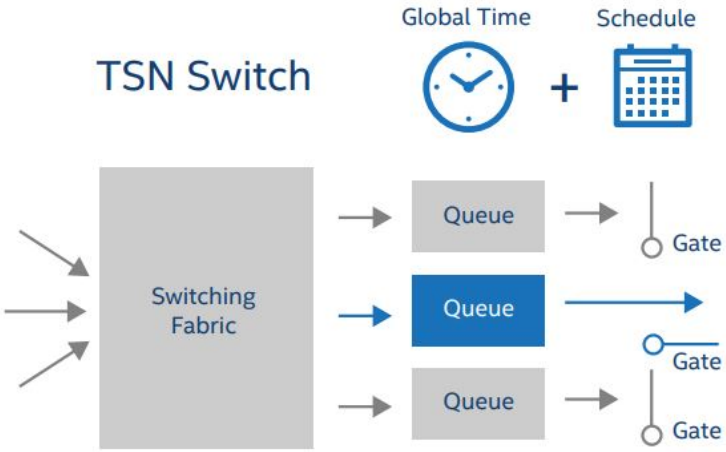

Figure 2.8: TSN Packet Forwarding with IEEE 802.1AS and IEEE 802.1Qbv

The major drawback of TSN that it is not designed for low latency hard real time environments. IEEE 802.1 does not guarantee latency lower than 2 ms [16]. Apart from this, it depends on switches to operate introducing additional delays and SPoF. Yet, it is one of the best among Industrial Ethernet solutions in terms of the capability of timing, synchronization, forwarding, queuing and redundancy all-in-one with IEEE 802.1 extensions, which gives inspiration to the novel protocol designed in this thesis.

2.3.4 Avionics Full-Duplex Switched Ethernet

Avionics Full Duplex Switched Ethernet (AFDX) [2], is derived from IEEE 802.3 Ethernet medium access control by adding timing and redundancy management on top of it. It enables deterministic communication by defining virtual links identities (VL IDs) and deploying switches to control the traffic. It has error detection, integrity check and dual redundancy. The redundancy is not only with physical links but also with redundant switches. Each frame is transmitted via two independent paths to the

redundant switches. The receiving end system discards the redundant frame, figuring out from repeated sequence number. Thanks to the dual redundancy, the protocol is tolerant to message lost unless the lost occurs in both channels. AFDX switches are responsible for preventing babbling of end systems and routing the packets based on VL IDs. Incidentally, end systems are the special names of network nodes in AFDX. Although it is possible to replace any failed end system, it is not applicable to add a new end system during operation since it requires update in the routing table of VL IDs.

AFDX has traffic shaping capability by defining virtual links. Once an AFDX switch receives a frame, it stores and forwards it based on the quality of service requirements. Each virtual link is unique to a source end system but can have one or more destination end systems [17]. VL IDs are embedded in the destination MAC addresses of Ethernet frames so AFDX has no special frame format. Minimum AFDX Ethernet frame is demonstrated in Fig. 2.9 cited from [18], containing UDP-IP network protocols as standard. Hence, the frame involves 47 bytes overhead which results in less efficient frame compared to overhead of raw Ethernet protocols, 18 bytes. The figure excludes preamble, start of frame delimiter and interframe gap fields.

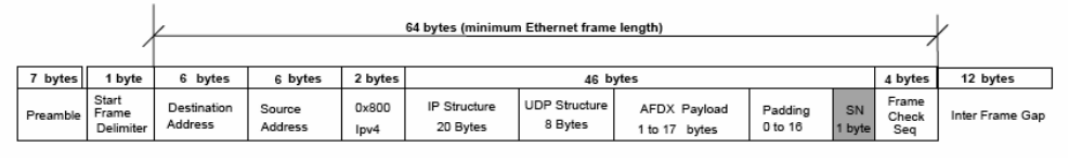

Figure 2.9: AFDX Frame Format

AFDX switches and end systems manage traffic shaping based on bandwidth allocation gaps (BAG) defined to each VL. The value of BAG indicates the minimum allowed time interval between two consecutive frames of the same VL ID [6]. It can be from 1 ms to 128 ms and must be a power of 2. An end system trying to transmit successive packets with the same VL ID in a smaller interval than BAG causes the latter to get blocked and queued for the duration of BAG. Unless the queue is non-empty, the upcoming packets with the same VL ID are also queued. Hence, end systems have a distinct queue for each VL ID they transmitting. Fig. 2.10 from [6]

demonstrates the traffic regulation under BAG condition.

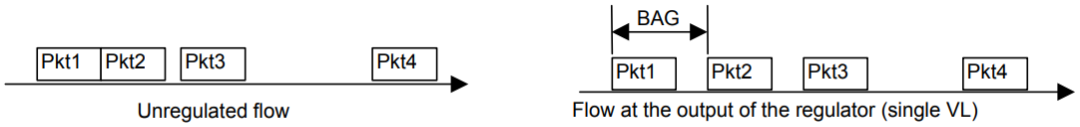

Figure 2.10: AFDX Flow Regulation with BAG

As end systems BAG control is shaping the traffic, there is also policing traffic capability in AFDX switches. It is token bucket based shaping where packets are dropped if there is no adequate credit for that VL ID. The account credit (AC) increases as time passes with rate (2.3.1). It is set to the value in (2.3.2) at startup and this is the maximum allowed credit for each VL ID. An incoming packet is forwarded to destination port, if there is eligible credit and the amount of credit equal to its frame size is cut. Insufficient credit causes packet drops.

$$AC_i^{rate} = \frac{S_i^{max}}{BAG_i} \tag{2.3.1}$$

$$AC_i^{max} = S_i^{max} \cdot (1 + \frac{Jitter_i}{BAG_i}) \tag{2.3.2}$$

$$40\,\mu s \leq Jitter_i \leq 500\,\mu s \tag{2.3.3}$$

where:

S_i^{max} = Preamble + Start Frame Delimiter + L_i^{max} + Interframe Gap

L_i^{max} = maximum frame size for VL i

As shown in (2.3.2), the jitter plays a role in the calculation of the maximum credit. AFDX end systems introduce some jitter bounded with (2.3.3) according to specifications. The jitter is due to the AFDX transmission technology and traffic shaping in end systems. The packet flow under jitter effect is represented in the following Fig. 2.11 cited from [6].

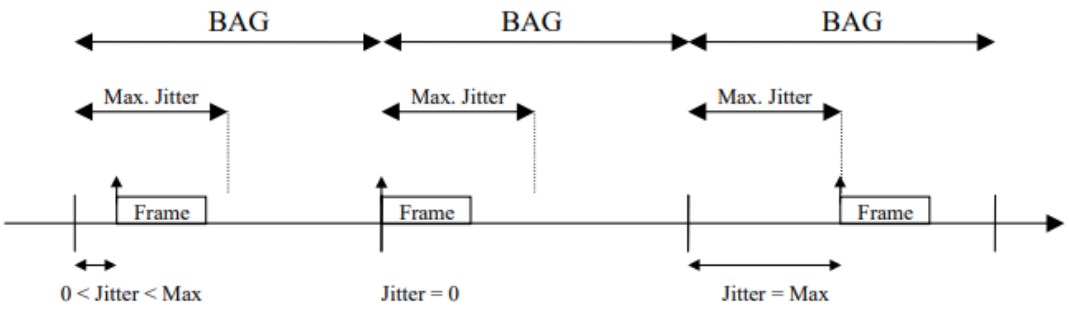

Figure 2.11: The Jitter Effect on AFDX Flow Regulation

The main disadvantage of AFDX is its strong dependency on switches, introducing more jitter and delay. The standard states that the latency in end system during transmission could be up to 150 μs. The total latency proportionally increases with number of links and number of switches on virtual link path [17]. In the scope of the thesis, AFDX is selected as the primary comparison protocol for the proposed protocol since it provides fast and reliable communication on Ethernet. Plus, there is an OMNeT++ model for AFDX. Finally, Chapter 4 includes the details of model provided by OMNeT++ and the simulation performed in Chapter 4.2.3 with real case traffic set.

Our literature survey shows that the existing protocols do not fulfill all the requirements of the network architectures for the real-time embedded systems that we state in Chapter 1.

CHAPTER 3

SHARED QUEUE BASED DYNAMIC SLOT RESERVATION PROTOCOL

In the conclusion of the paper "Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet" [19], it is highlighted that to share the benefits of IEEE 802.1 AVB and time-triggered communication, an interesting subject of future work is an in-car network that uses both protocols on the same physical layer, using time-triggered messages for critical control data and event triggered messages for time sensitive streams. In the scope of the thesis, the proposed protocol comes up with a solution to handle both time triggered and event triggered packets in a collision free shared medium. The protocol solves the problem with its broadcasting communication architecture. This enables nodes to gather relevant network information to handle scheduling, fault management and synchronization.

3.1 Motivating Specifications

Shared Queue based Dynamic Slot Reservation (SQDSR) protocol is designed for meeting the communication requirements of the fault tolerant hard real time embedded systems. It is developed by looking at the advantages and disadvantages of the prevalent communication architectures. The protocol is based on a half duplex shared bus with Ethernet physical layer since Ethernet is the prevalent technology thanks to its compatibility, low cost components and high speed capability. There are no any additional smart network components including switches that may introduce more delay and non determinism.

The proposed protocol is a combination of time triggered and event triggered communication architectures, having variable length slots where some of them are statically

assigned to nodes at the beginning of communication, called static slots, and some of them are left to dynamic allocation during run time, called dynamic slots. Static slots are used similar to the TDMA (Time Division Multiple Access) based approach to handle periodic packets while dynamic slots are reserved for sporadic packets to provide temporary slots. A repeated pattern of slots is called as cycle providing a modular design to the communication architecture. The cyclic process paves the way for scheduling and worst case analysis.

The main principle of the architecture is broadcast communication over the bus. The broadcasting communication is a must for the basic working principle of the protocol; shared queue based medium access. Shared queue is an internal queue that each node maintains in order to decide on the allocation of dynamic slots. The queue contains records of information on the pending sporadic packets notified by each network component. The notification is realized by inserting the information into the Ethernet frame payload. The information contains the record for the top packet waiting in the sporadic queue. Since all the nodes listen to the shared bus, they are able to update their shared queue with the latest information. The arbitration of the bus access is resolved through the shared queue so that the node possessing the highest priority packet on the shared queue wins the arbitration and has the shared medium access in the next dynamic slot. Note that each node maintains exactly the same copy of the shared queue thanks to the broadcast communication.

The broadcasting nature of the SQDSR protocol makes fault tracking and handling easy. The nodes can detect any failure and react accordingly. Most of the prevalent protocols possess only a single node which is aware of the fault conditions. This introduces the single point of failure problem. Plus, remote nodes are not capable of fault reporting. Since the proposed protocol is designed for fault tolerant real time embedded systems, the way of fault management and reporting is an essential point of the design.

3.2 Overview

The proposed protocol is constructed in SQDSR Controller as a link layer protocol on top of Ethernet MAC layer as shown in Figure 3.1. It is designed to be implemented on hardware to reduce variable processing delays based on the operating system. SQDSR protocol has a predefined frame format explained in Section 3.4 with special inner header inserted into the beginning of the Ethernet payload field. The header contains message ID information to indicate whether it is a control, periodic or sporadic message. This field is filled by the application layer software; control traffic generator, periodic traffic generator and sporadic traffic generator. The message type parameter is used to classify packets in the SQDSR protocol layer in which the packets are stored in classified queues and handled differently.

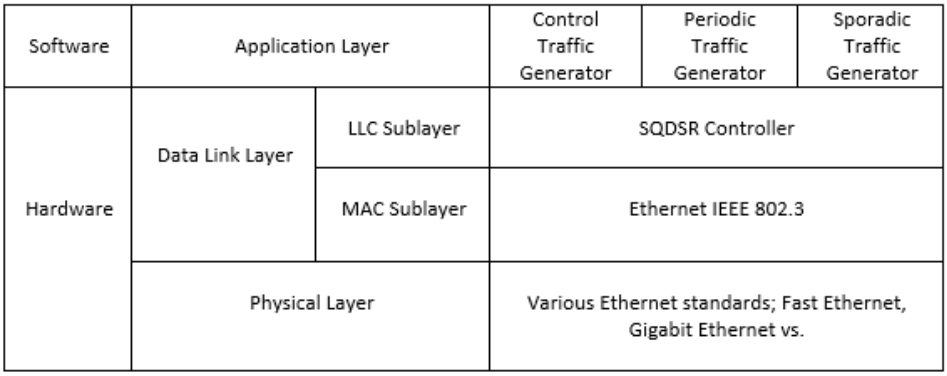

Figure 3.1: SQDSR Protocol Stack

SQDSR protocol communication architecture is formed by three network states defined as control state, static state and dynamic state. The states differ from their slot characteristics. In the scope of the thesis, the slot concept is used as a time interval where only one node is able to access the shared bus and allowed to send merely one Ethernet packet during this interval. It does not have to be a fixed time interval but its length is naturally restricted by the maximum Ethernet frame length. Since all the network components listen to the bus consistently, they know the end of each slot through the end of frame. The slots are named based on the state they belong to. This naming convention creates control slots, static slots and dynamic slots. The slot at the beginning of the dynamic state is the retransmission slot which is used to notify the

faulty packets that need to be retransmitted.

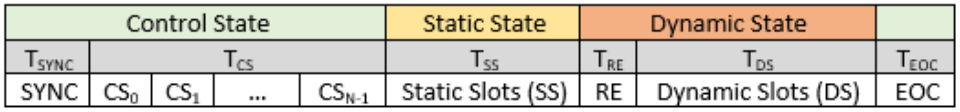

Figure 3.2: SQDSR High Level Cycle

Fig. 3.2 demonstrates a high level cycle of the protocol. It is the cycle where there is a control state at the beginning. The first row indicates the state of the network and the second row is for time intervals of the corresponding slots. The cycle starts with the control state that is responsible for synchronization and information exchange between nodes. Next, the static state follows containing periodic slots scheduled in advance. The end of the cycle is left for the dynamic state to provide nodes to reserve additional slots as demanded during run time.

Having control slots at the beginning of each cycle usually ends up with a poor effective data utilization. Therefore, the protocol is designed to have the control slots only in each previously determined period of cycles. This period is referred to the high level cycle period. The remaining cycles without control slots are named as low level cycles. The low level cycles are available just after the high level cycle and repeated until the next high level cycle; then, the whole process restarts again. The following figure is a representation of this process where CS, SS, RE and DS refer to control slots, static slots, retransmission slots and dynamic slots respectively.

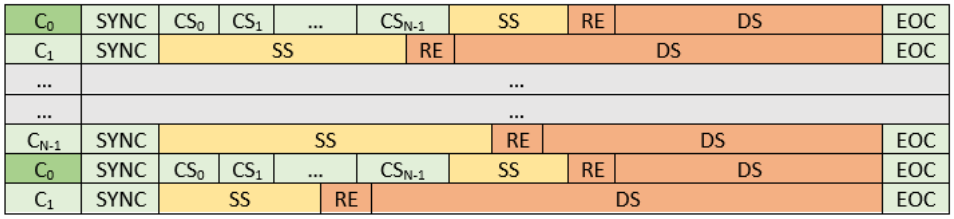

Figure 3.3: Illustration of SQDSR Successive High and Low Level Cycles

The further sections clarifies the algorithm behind each network state with the timing analysis of slots. Next, the state transitions are explained with flowcharts. The ap-

plied process for time synchronization between the nodes follows the state transition section. Finally, the frame format of SQDSR protocol is going to be discussed.

3.2.1 Control State

At the beginning of each cycle, the network state is set to the control state, where the time synchronization and notifications of status take place. The Figure 3.4 represents the slot timing structure for the high level cycle control state. The state starts with the synchronization slot (SYNC) indicating the start of a new cycle where the synchronization frame is transmitted. The frame is sent by a specific node determined beforehand, called time synchronization master who is responsible for synchronizing the clocks for the rest of the nodes. Likewise the start of cycle slot, the synchronization slot, there is also an end of cycle slot, abbreviated as EOC. It is an interval left empty to keep communication idle until the next cycle in order to keep the nodes prepared to the next cycle.

The name of state is coming from the successive notification of control information from the nodes. Each control slot is assigned to one unique node determined in the design time. Minimum length Ethernet frame is transmitted in each control slot, that consists of information on status and demands of the node. Each node notifies the status about the waiting packet on top of the sporadic queue based on the priority. If the queue is non empty, then the node requests slot allocation in dynamic state. The rest of the nodes processes the notification and updates their shared queue with this new record. This shared queue is going to be used in the arbitration process at dynamic state slots. The content of the records and how they transmitted are detailed in the further Section 3.4. Apart from the control information, control slots are part of the time synchronization process explained in 3.3.

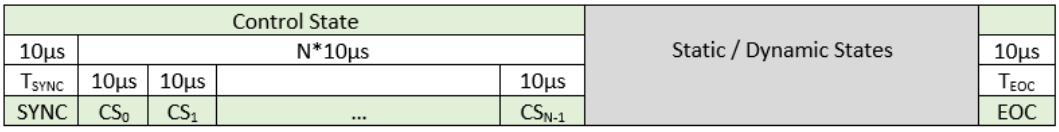

Figure 3.4: SQDSR Control Slots

One of the major concern with the control slots is the specification of the slot length. These slots carry small information that can fit into the minimum Ethernet frame with 46 bytes payload. This frame takes up to total 84 bytes including Ethernet header, preamble, start frame delimiter and inter-frame gap. The transmission of this frame takes 6.72 μs in case of 100 Mbps bit rate. Therefore, the length of the slot should be larger than this value. Note that these slots are time triggered, thus there is no need to process neither the transmitted nor received packet. The packet to be transmitted should have been ready at that time instant. Therefore, the processing delay is only the time passed between network interface driver and the physical layer. This delay is measured approximately 1 μs in [20]. Plus, there could be time synchronization errors results in lagging. The synchronization is accurate under 1 μs for IEEE 1588 Precision Time Protocol with hardware time stamping according to the paper [21]. As a result, it is decided to determine length of the control slots as 10 μs.

Moreover, control slots are alive or fault checks for the components in the network. For instance, they could be used to verify the distributed shared queue information throughout the nodes. It is possible to insert a part of the shared queue into the control packet so that the information can be checked by the other components to ensure all have the same records in their shared queues. Although it is not expected to have distinct shared queue records, the example was given to illustrate a possible scenario that control slots could be used to prevent any faults.

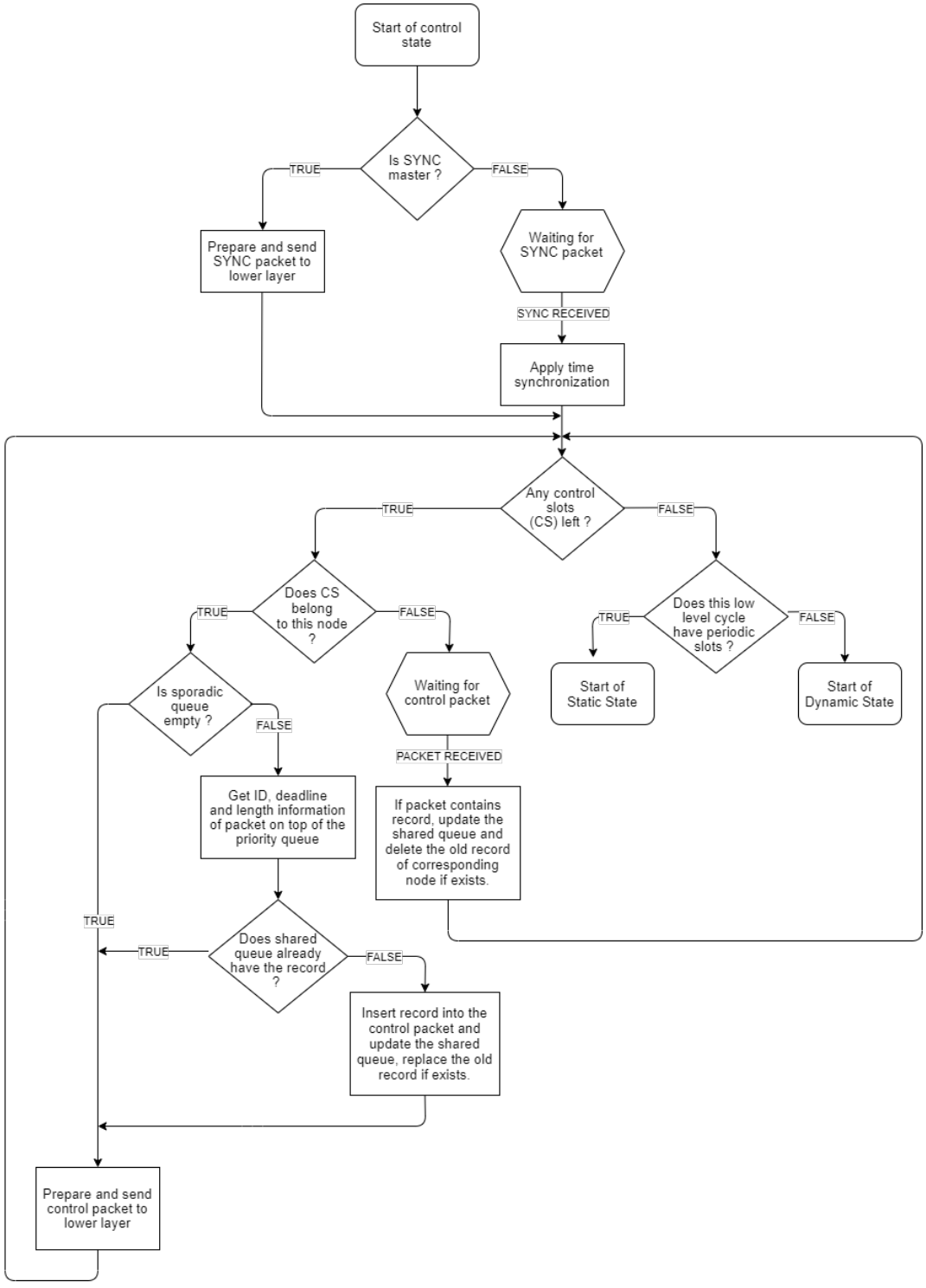

Figure 3.5: Flowchart of SQDSR Control State

3.2.2 Static State

SQDSR protocol supports TDMA likewise slotted architecture and the static state is specialized for it. It is available in each cycle as far as there is at least one scheduled static slot. Static slots are shared medium access form and determined prior to the run time. The slots are assigned to the nodes having periodic messages to transmit. Since all of the network components know the designed schedule, the one who's going to transmit next is evident. Probably the biggest difference compared to TDMA is that the length of static slots is not fixed. Thanks to the broadcast communication, the successive node finds out when the previous slot ends with the EOF signal. This paves the way of efficient slot utilization for distinct length of frames.

The interval of static state depends on the load and distribution of periodic messages. A cycle could contain no assigned slot, thus no static state, which leads to skip directly to the dynamic state. It is allowed to have more than one periodic slot per node for each cycle but there could solely one packet per slot to be transmitted. An important design point on scheduling static slots is to consider the time left till the end of the low level cycle. This is the maximum allowed period to be utilized for static state. High utilization of static slots leads to lack of time for dynamic slots.

Even if there is no periodic packet to send, the components should notify it with a message in the dedicated slot. Not transmitting anything, raises a fault in the network and should be handled properly in the re-transmission part. The fault is realized when the maximum allowed time to transmit is lapsed. The duration of 10 μs for timeout is selected by relying on the measurement in [20]. The Figure 3.6 represents a static state with three different length slots and the maximum permitted guard time of 10 μs.

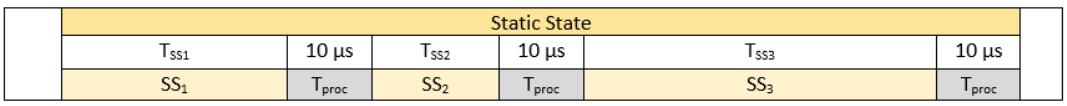

Figure 3.6: SQDSR Static Slots

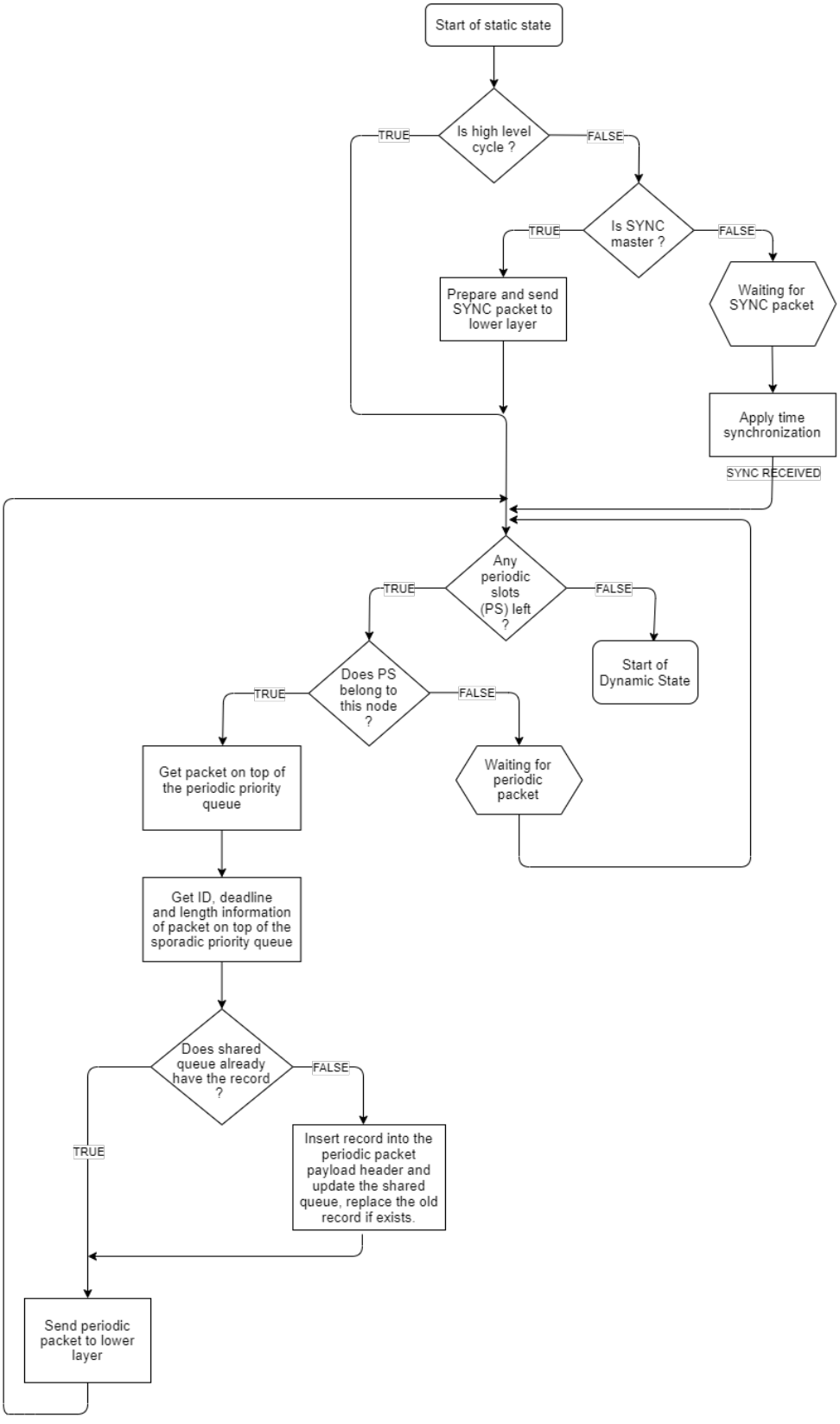

Figure 3.7: Flowchart of SQDSR Static State

3.2.3 Dynamic State

Dynamic state is the last part of the SQDSR cycle where slots are dynamically allocated based on the shared queue records. Dynamic slots are free space to utilize the remaining time left from static slots in order to handle time sensitive sporadic data. It is a special interval to serve packets of any node without waiting for the entire communication cycle.

The slot reservation process is enabled not only in control slots, but also static and dynamic slots with piggybacked reservation record in payload. The details of the record is explained in the further Section 3.4 SQDSR Frame. In short, it contains the record of the highest priority packet waiting in the queue and ready to transmit. The notification of records updates the shared queue of each node. The records are ordered by their deadlines because the dynamic state exist for handling time critical packets before missing the deadlines.

The start of the state begins with the re-transmission message (RE) sent by the predetermined master selected at design time. This frame contains information on any fault occurred in the previous cycle. In this case, faulty messages should be re-transmitted in the dynamic state. The frame format of re-transmission message is discussed in the Section 3.4, but briefly, it announces faulty message IDs from the previous cycle. Just after the re-transmission message, erroneous packets are re-transmitted in the order they announced in the notified message. Hopefully, the re-transmission processes do not have to be used often so that the communication proceeds with dynamic slots.

A new record could change the ordering in the shared queue. Each dynamic slot could affect directly the next slot arbitration process. The shared queue records might also be updated in dynamic slots, therefore, who's going to transmit in the next slot is not known till the previous frame received and processed. Moreover, if the re-transmission master implies any fault, the next slots are allocated for fault management rather than dynamic slots.

The arbitration process is mostly straightforward. The node having the highest priority record, i.e. the record on top of the queue, selected to gain the access of bus. However, the frame should be checked whether it fits to transmit till the end of the

cycle or not. If not, the next record is investigated for the same constraint. If none of the records are appropriate, the remaining time stays unallocated.

Arbitration process in dynamic slots results in a delay on start of each slot. The source of the delay is the processing time to receive previous frame, process it, update the shared queue, decide on the arbitration and transmit if the record belongs to itself. If the node is not transmitted even if the record belongs to it, then it is a fault condition needs to be handled in the next cycle re-transmission part. For this reason, there should be a predefined guard time to indicate the limits of transmission lag. It is designed as 15 μs referring the paper [22] proving this amount of guard time is sufficient to separate slots in a similar communication architecture.

The next figures show two different dynamic state schedule instances. The former, Figure 3.8, does not have any faulty message re-transmission and the dynamic slots start just after the re-transmission message. This cycle is probably not highly loaded or waiting packets are too long to fit, since the end of the dynamic state is left unallocated. The latter, Figure 3.9, has one faulty message from previous cycle, that's why dynamic slots lags until the completion of re-transmission.

	Dynamic State							
	T_{RE}	T_{DS1}	15 µs	T_{DS2}	15 µs	T_{DS3}	$T_{Unallocated}$	10 µs
	RE	DS_1	T_{proc}	DS_2	T_{proc}	DS_3	Unallocated	EOC

Figure 3.8: SQDSR Dynamic State with No Re-transmission Required

	Dynamic State								
	T_{RE}	T_{CS}	15 µs	T_{DS1}	15 µs	T_{DS2}	15 µs	T_{DS3}	10 µs
	RE	R_{CSX}	T_{proc}	DS_1	T_{proc}	DS_2	T_{proc}	DS_3	EOC

Figure 3.9: SQDSR Dynamic State with A Re-transmission of Faulty Frame

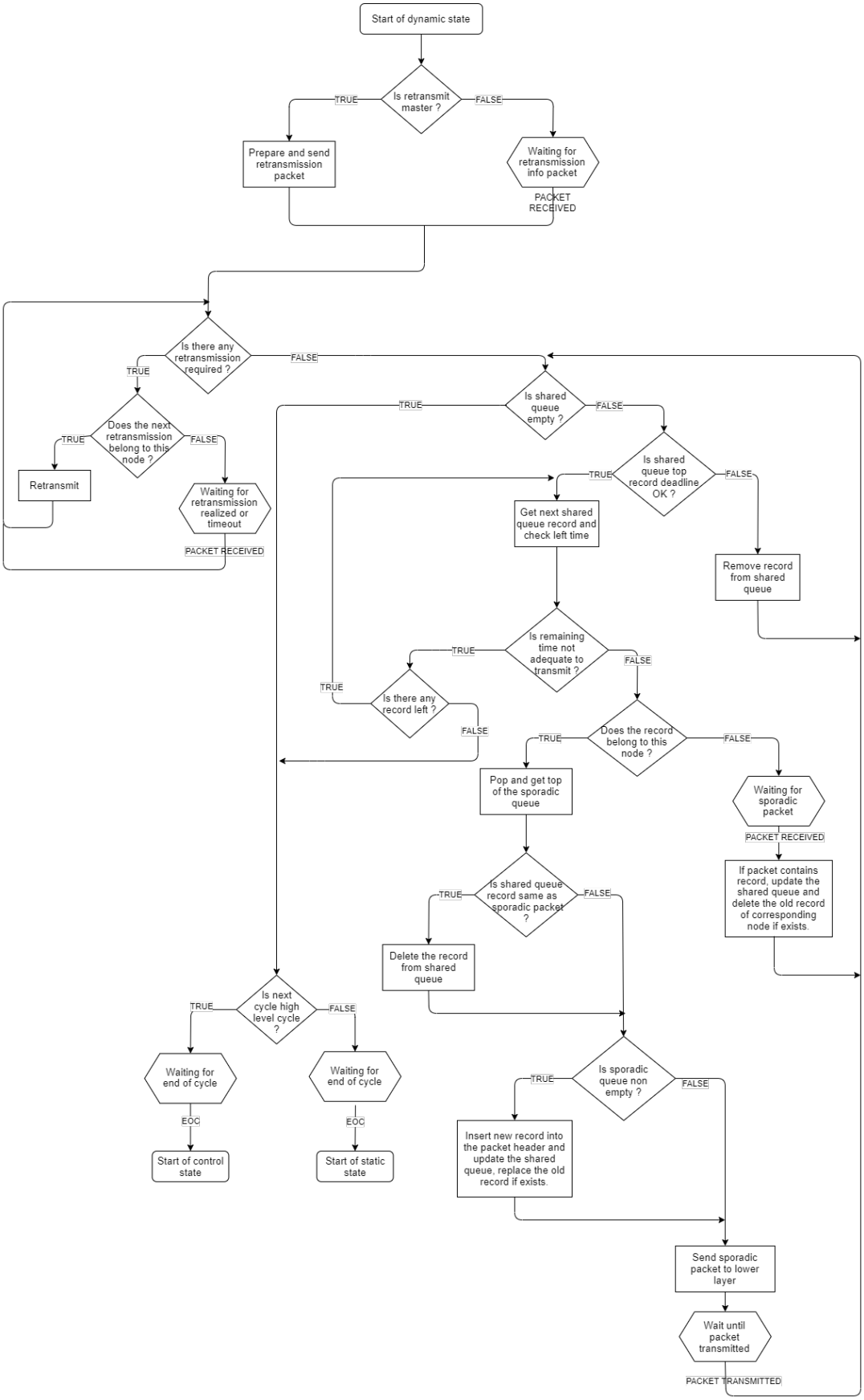

Figure 3.10: Flowchart of SQDSR Dynamic State

3.3 Time Synchronization

Synchronization among nodes plays an important role for time sensitive applications. Time slotted networks require accurate clock synchronization for reliable communication. The synchronization is the key component for time triggered architecture of control slots for SQDSR protocol. Since the control slots carry Ethernet frame with minimum length, it is not efficient to implement the network without precise clock synchronization. It takes only 6.72 μs for transmission of 84 bytes Ethernet frame including preamble, start of frame delimiter and inter frame gap over 100 Mbps physical layer. Besides, the react of network interface card, i.e. round trip latency between receive and transmit of a frame, takes similar amount of time [20]. Therefore, the nodes should be able to transmit in the exact time of assigned control slots without waiting the completion of processing previously received packets. In this way, it ends up with minimum processing delay making possible to fit a control slot.

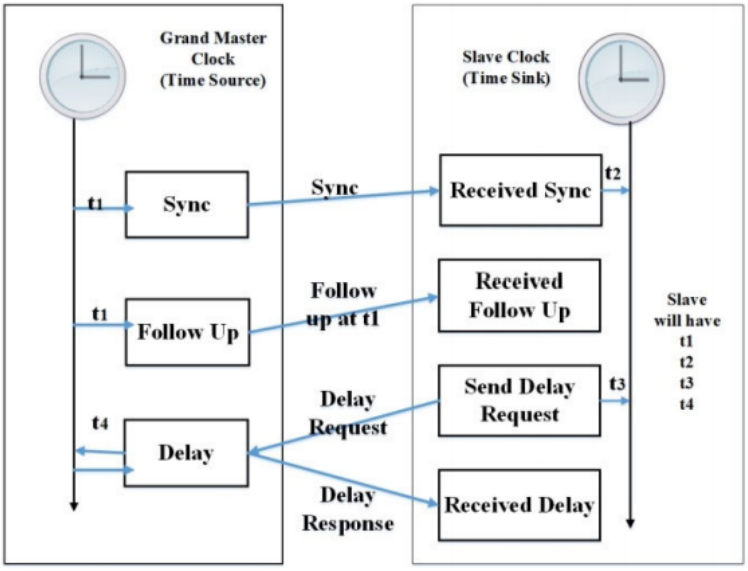

Figure 3.11: IEEE1588 PTP Messages to Exchange Clock Information

Even if the clocks of nodes are adjusted at a specific time instant, they tend to drift separately as time forwards. The oscillator distinctions and environmental effects are major reasons for this clock drift. The solution to the problem is provided through structured implementation of Precision Time Protocol based on IEEE 1588 [21], by

exchanging messages between one clock master and multiple slaves as shown below. The synchronization with IEEE 1588 Precision Time Protocol has under 1 μs accuracy based on the article [21].

IEEE1588 PTP has predefined messages to manage time synchronization as described in Figure 3.11 cited from [21]. It requires one master node and multiple slave nodes with the intent of synchronizing slave clocks with the clock of master. The algorithm starts with `SYNC` message sent by master to slave. `FOLLOW UP` comes after it, carrying the timestamp info when `SYNC` was transmitted. Aftermath of `SYNC` and `FOLLOW UP`, each slave node sends `DELAY REQUEST` message to the master and expects the timestamp when it is received. The reception time is sent with the response message `DELAY RESPONSE` . By combining all available timestamps, there is enough information to find out transmission delay between nodes, therefore, the offset between slave and master, using the following equations (3.3.1), (3.3.2) and (3.3.3).

$$Offset_{slave} + Delay_{transmission} = t_2 - t_1 \tag{3.3.1}$$

$$Delay_{transmission} - Offset_{slave} = t_4 - t_3 \tag{3.3.2}$$

$$Offset_{transmission} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{3.3.3}$$

SQDSR protocol handles the exchange of time synchronization related information in control slots. As required by IEEE 1588 protocol, one master node should be chosen. The master does not need to have any specific ability, thus any node can be appropriate. For the sake of fault tolerance, a secondary master should exist to replace in any erroneous condition. Time synchronization master sends `SYNC` message in the SYNC slot of control state and the `FOLLOW UP` message in the control slot assigned to the master node. Slave nodes send `DELAY REQUEST` in control slots assigned to each of them so that `SYNC`, `FOLLOW UP` and `DELAY REQUEST` messages are done in a single cycle. The remaining `DELAY RESPONSE` is transmitted in the SYNC slot

of the next cycle from the synchronization master. Note that the communication is always broadcast in SQDSR and it is only allowed to send one frame per slot.

The Figure 3.12 represents the layout of IEEE1588 messages in control slots. Slaves calculate the offset using equation (3.3.3) and correct their clocks. In this way, the process is completed in two successive cycles.

Control State							Static / Dynamic States	
SYNC	CS_0	CS_1	...	CS_{MASTER}	...	CS_{N-1}		EOC
1588_{SYNC}	1588_{D_REQ}	1588_{D_REQ}	1588_{D_REQ}	1588_{FOLLOW_UP}	1588_{D_REQ}	1588_{D_REQ}		
1588_{D_RESP}								

Figure 3.12: IEEE1588 PTP Messages on SQDSR Slotted Structure

3.4 Frame Format

SQDSR protocol operates on Ethernet frames by means of inserting additional information into the Ethernet payload. The destination address of frames are mapped to broadcast MAC address `FF:FF:FF:FF:FF:FF` due to the broadcast communication architecture of SQDSR, and the source address field indicates the source address of node transmitting the frame. Ethernet header type field is assigned to specific value to show that it is a SQDSR frame. The SQDSR protocol uses Ethernet payload to piggyback SQDSR protocol headers as seen in Figure 3.13, which are message ID of the frame, data length and shared record information. Since there is an additional field after the data field of SQDSR specialized payload, the length of the data should be specified explicitly.

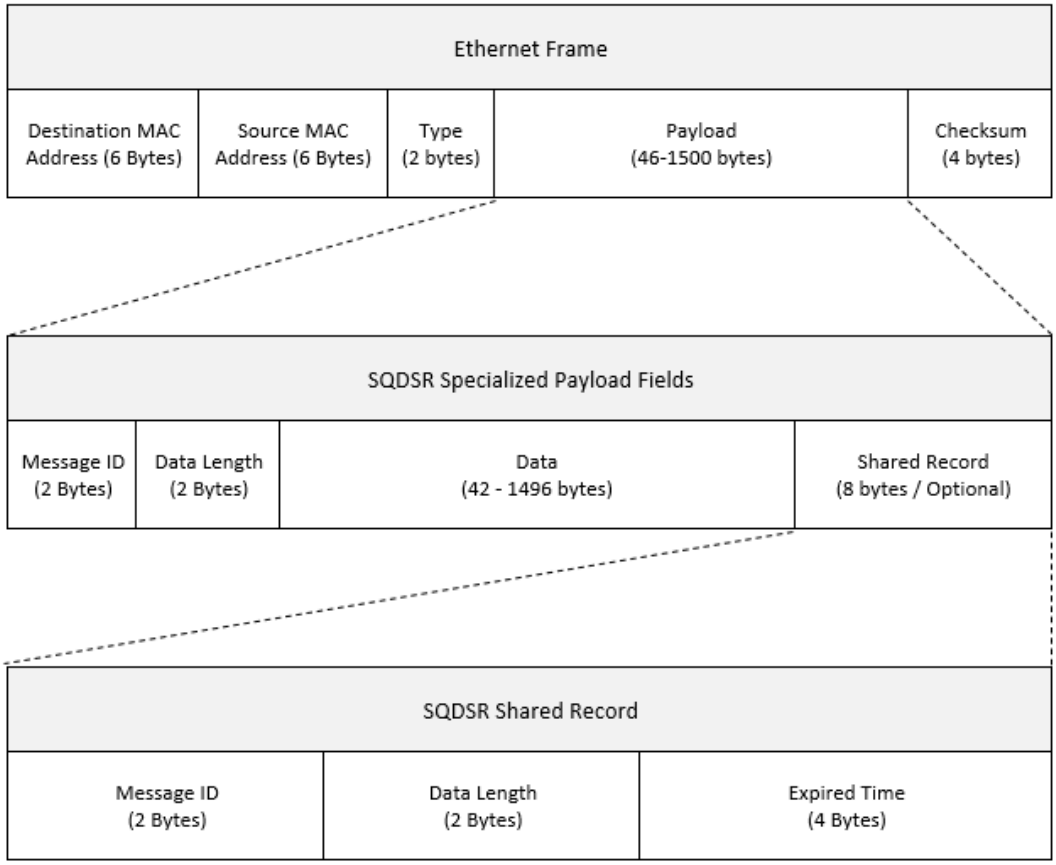

Figure 3.13: SQDSR Ethernet Frame Format with Piggybacked Shared Record Information

Message ID field stores either reserved unique number for some sort of slots or the transmitted packet message ID for static and dynamic slots. The reserved message IDs are shown in the Table 3.1. These are for the internal operation of the SQDSR protocol. The reserved message IDs are restricted for use by the user defined messages. The frames with reserved message IDs consist of IEEE 1588 time synchronization messages, re-transmission message and control messages.

Shared queue record is piggybacked to the data field of Ethernet payload in order to notify the top waiting packet in the sporadic queue. It is allowed to notify a single record per Ethernet frame. The record includes message ID, data length and expired time. Other nodes get the record, obtain the required information of the notified packet and save it to the shared queue. These records are deployed for the arbitration process in dynamic state. Expired time information has four bytes to store the time

Reserved Message IDs	Description
0xFFFF	Re-transmission Frame
0xFFFE	Standard Control Frame
0xFFFD	IEEE 1588 SYNC Frame
0xFFFC	IEEE 1588 FOLLOW UP Frame
0xFFFB	IEEE 1588 DELAY REQUEST Frame
0xFFFA	IEEE 1588 DELAY RESPONSE Frame

Table 3.1: SQDSR Reserved Message IDs

instant when the message misses its deadline. Every time information in SQDSR frames is in the microseconds range.

Re-transmission frame is a special packet transmitted in each cycle at the beginning of dynamic state. The frame notifies the faulty messages of the previous cycle. The erroneous packets are successively declared with the pair of message ID and source ID. The responsible units should retransmit the packets with respect to the notification ordering. The representation of the re-transmission frame is shown in Figure 3.14. The message ID of the frame is set to `0xFFFF` and the data length set to multiplication of number of faulty packets with three bytes.

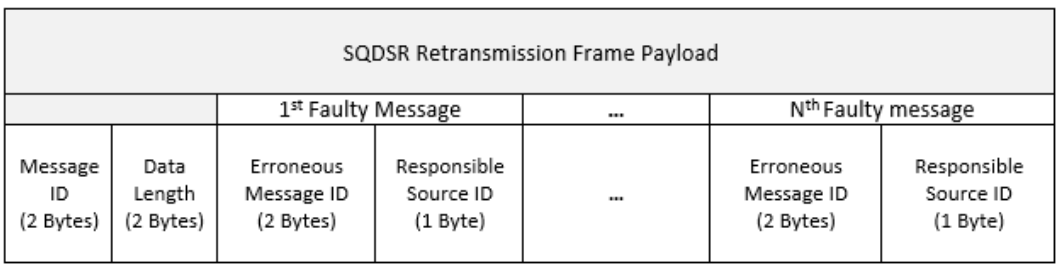

Figure 3.14: SQDSR Retransmission Notice Frame for Faulty Packets

3.5 Fault Tolerancy

The primary design requirement of SQDSR protocol is to provide fault tolerant communication architecture. The mechanisms and techniques that enable the system deliver consistent service even in the presence of fault is called Fault Tolerance (FT) [5]. The way of implementing FT is realized by redundancy management. Redundancy is a backup mechanism that could be applied to either hardware or software. Replicating the same procedure with backup hardware, software and also with the same hardware and software in another time is the implementation of redundancy in order to ensure the correct output of the operation.

SQDSR protocol has designed on the base of broadcast communication which enables to track and report any fault. Every node monitors the traffic flow consistently and could detect any problem instantly. There are mainly two types of traffic flow; statically scheduled flows at design time and dynamically scheduled flows at run time. The previously scheduled traffic flows are checked whether they are consistent or not by looking at the schedule table distributed to nodes in advance. Whereas dynamically arranged traffic is controlled with the help of shared queue that all nodes have the same exact copy. The shared queue mechanism enables common and synchronized arbitration for dynamic slots in which any incorrect medium access raises an exception. Thus, monitoring the entire traffic thanks to the broadcast communication makes the fault awareness possible. Plus, it provides to report any failure from any node to the client simultaneously.

Even though there is no master node to control the network in SQDSR, there are still some nodes having additional tasks such as time synchronization and retransmission. Having a special node with critical responsibility induces SPoF issue. There are many protocols suffer from SPoF probability and tackle the problem with hardware redundancy. The same manner is applicable for time synchronization and retransmission masters of SQDSR. One advantage of SQDSR protocol, the said masters have no particular feature that's why they are easily replaceable with other nodes. The secondary master nodes are substituted if any misbehaviour is detected. Hence, the backup nodes should be configured to the system in advance.

The fault handling mechanism of SQDSR relies on the retransmission slot. This slot is available in each cycle at the beginning of dynamic state. The retransmission master declares any erroneous frames from the previous cycle with their message and responsible unit IDs. An example of retransmission frame is demonstrated in Figure 3.14. After the retransmission slot, the faulty frames are retransmitted again with respect to the ordering in the notification frame. If there is no fault record for that moment, the retransmission slot has zero length data field and the state continues with dynamic slots.

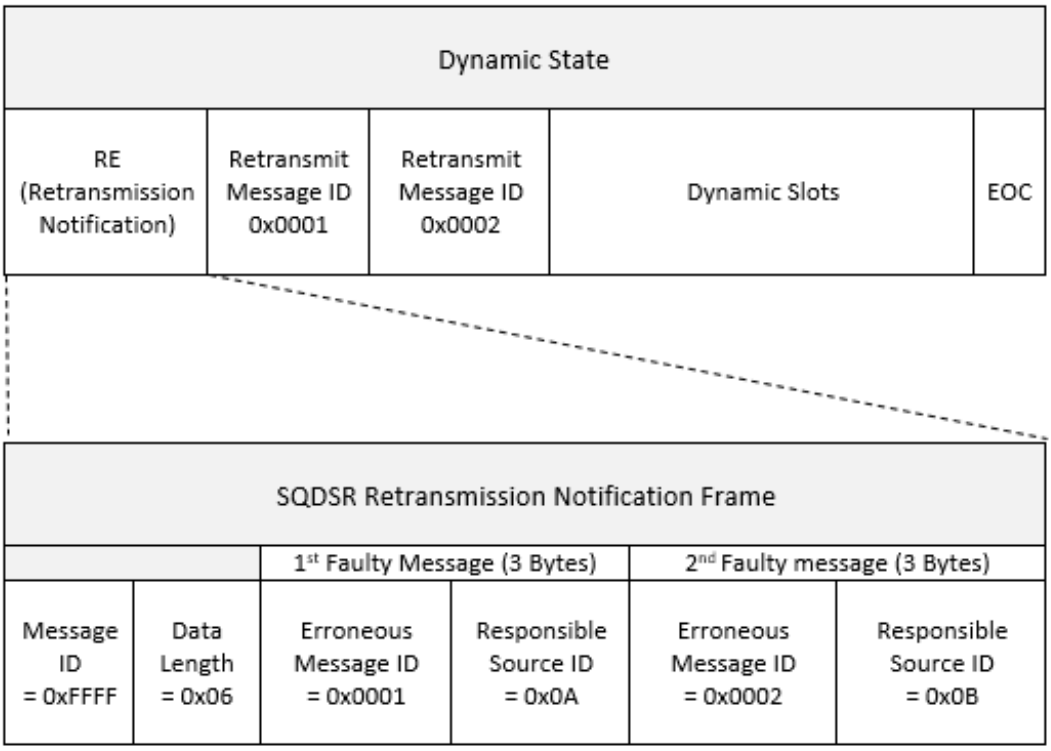

Figure 3.15: Illustration of SQDSR Retransmission with Two Faulty Messages

Figure 3.15 illustrates a simple scenario of fault handling mechanism for two erroneous packets. The retransmission master declares that frames with message ID 0x0001 from node 0x0A and message ID 0x0002 from node 0x0B have not transmitted successfully. Aftermath of retransmission notification, the responsible source nodes need to retransmit corresponding frames respectively. If the failure keeps going for the retransmitted frame, the source node is marked as faultier and should be diagnosed.

CHAPTER 4

PERFORMANCE EVALUATION

It is important to demonstrate the performance of proposed protocol SQDSR on realistic network traffic. This section provides a complete network simulation to understand capability of the protocol. Average and maximum network delay, queuing length, schedulability of the given message set are some of the investigated performance metrics in the scope of thesis. The simulation is performed on OMNeT++ which provides built-in network layers and real world simulation environment.

It is rational to compare the proposed protocol with one of the prevalent real time communication architecture. AFDX protocol is selected for this purpose since it provides Ethernet based deterministic communication via virtual links and redundancy for any faulty condition thanks to its switched architecture. Most importantly, OMNeT++ supports AFDX simulation model from [23] so that the comparison could be realized under similar parameters.

4.1 Evaluation Criteria

The way of performing simulation could be more importantly than the results of it. The simulated network should be rational and schedulable. Plus, the results should be reliable and repeatable. Therefore, OMNeT++, an official network simulator framework, is used with the real traffic data obtained from a commercial company. The designed network is tested for reliability and schedulability. The results containing latency measurements are validated with the confidence interval 4.1.3.

4.1.1 Latency Modelling

The evaluation of simulation results rely on latency measurements for the compared protocols. The latency refers to the amount of delay the packets experience from the generation time at source to the reception time at destination. It is application to application delay including processing time of packets in hosts, propagation time on physical medium from source to destination and the protocol delay due to slot timing and queuing. The simulations are performed under 100 Mbps physical bit rate, so the transmission delay for each packet can be calculated with the following equation (4.1.1).

$$T_{transmit} = \frac{FrameLength_{bits}}{100Mbps} \tag{4.1.1}$$

Since we are comparing two different protocols; one with shared medium and another with switched architecture, the latency model can be quite different for them. The shared medium based architectures have single transmission whereas the switched architectures have at least two transmission for single message from source to destination. Including the processing delay occurred in switches, the latency model consists of the following parameters in (4.1.2) excluding the transmission delay. The processing delay components are constant and known at design time, hence even if the latency results involve these components, it is possible to remove them.

$$T_{latency} = Tproc_{source} + T_{propagation} + n \cdot (Tproc_{switch} + T_{propagation}) + Tproc_{destination} \tag{4.1.2}$$

4.1.2 Reliability and Scheduling

Simulation results are only meaningful in a fault tolerant real time system if none of the packets are lost or dropped. Therefore, the main criteria during design is to achieve schedulable and reliable network. To this end, both protocols are tested un-

der the given traffic load whether or not the network manages all the generated packets. The maximum amount of delays introduced are followed to check for deadline conditions.

Next, the simulation results need to be controlled to verify outputs. The rationality of measurements should be checked manually. The states of each network components are controlled during simulation. The models in simulation framework is designed as safe such that any missing information, dropping packet or unexpected behavioral raises an exception. Therefore, the obtained results could be assumed to be safe under these constraints.

4.1.3 Confidence Interval

One of the crucial parameter when performing a simulation is to make sure on collecting the adequate number of results. Since it is not possible to perform the simulation forever, we never know the true mean μ. However, it is important to know how close our sample mean is y_n with respect to true mean. It is likely to say our sample average y_n is within $\triangle\%$ of the true mean μ with a confidence level of g% that we would have if n goes to infinite. Using the following equation, where s_n is standard deviation, n is number of samples and t_g is the constant selected 1.96 or 2.58 for g 95% and 99% respectively [24]. All the simulation results for both SQDSR and AFDX are verified by the confidence interval of true mean. The equation justifies that simulation outputs are below 1% of the true mean with probability of 99%. The results are obtained with 10 seconds of simulation for the given network traffic which generates 128.000 packets at the end.

$$\triangle\% = \frac{s_n \cdot t_g}{y_n \cdot \sqrt{n}} \tag{4.1.3}$$

4.2 Performing Simulation with OMNeT++

OMNeT++ [4] is an open source, discrete network simulation framework based on C++ and distributed under Academic Public License. Components are modeled in

Network Description Files (NED) that provide to create more complex modules by connecting and assembling them. It supports modelling wired and wireless networks, protocols, queuing; plus, analyzing and validating the developed model. OMNeT++ offers graphical run time environment and Eclipse based IDE.

INET framework [5] is an open source library for OMNeT++ containing prevalent Internet protocols and implementation of OSI network layers. Ethernet protocol support is available in INET framework, including Ethernet MAC layer, encapsulation of Ethernet packets and generating Ethernet traffic. Therefore, it is suitable to utilize INET framework for the simulation of SQDSR.

Although `NetworkSimulator` was developed and tested for various protocols during the research as in Section 2.2.1, OMNeT++ was ultimately preferred for testing and validating protocols since it is official and verified network simulation tool. In the scope of this thesis and based on the specification that we use as input for our design, the simulation models for AFDX and SQDSR are performed, tested and analyzed with OMNeT++ version 5.6.2 and INET version 4.1.2.

4.2.1 OMNeT++ Overview and Models

The model of the proposed protocol is constructed on top of `EtherHost` model of INET framework. `EtherHost` model is under `inet.node.ethernet` package of INET framework and designed as a simple Ethernet host with one Ethernet traffic generator and one Ethernet port to simulate request reply traffic between nodes. `EtherTrafGen` is under `inet.applications.ethernet` producing Ethernet frames.

Apart from the traffic generator, the model contains `MessageDispatcher` to connect multiple application layer protocols with each other and dispatch packets between them; `EtherEncap` to perform LLC/SNAP encapsulation and decapsulation of Ethernet frames; `IEthernetInterface` to simulate Ethernet MAC layer and physical layer properties. These modules are all used with the default parameters during simulations. `IEthernetInterface` model supports full duplex communication but it is disabled for SQDSR protocol. The promiscuous mode is also enabled

so that nodes receive all the packets in the shared medium regardless of the packets are destined to them. This enables to simulate the broadcasted nature of SQDSR. OMNeT++ gives the opportunity of creating new modules. The modules are able to transmit and receive frames with the interface provided by OMNeT++. Any implementation is possible including queuing or discarding packets, introducing delays and recording information about traffic. Therefore, it is not obligatory to depend strictly on INET models. New modules are used in the design of SQDSR in order to improve the functionality of original `EtherHost` model.

4.2.2 SQDSR OMNeT++ Model

`EtherHost` model of INET framework is enhanced to create generic model of SQDSR host as shown in Fig. 4.1. The most apparent modification to the `EtherHost` INET model is that the number of `EtherTrafGen` modules is increased to three to distinguish the traffic generators of control frames, periodic frames and sporadic frames. These separate instances are also designed as arrays of `EtherTrafGen` module with parametric count value. Since a traffic generator is designed to produce only one type of packet, multiple packet generations per node can only be accomplished with multiple traffic generators. Application interface layers following the traffic generators are new add-on components which are simple OMNeT++ models. Their main task is to keep the statistics of transmitted and received packets. There is also another add-on component, called Delay, that is not available in original `EtherHost` model but created to simulate processing delay in host machine. Finally, the last layer before Ethernet physical layer, `protocolLayer`, is created to implement SQDSR protocol operations.

Types of network components deployed to create SQDSR custom host are demonstrated in Fig. 4.2. Besides the instant components of INET, there are other components inherited from `SimpleModule` of OMNeT++ and designed from scratch. This module enables users to implement custom network components. It has an interface to receive and transmit from adjacent layers so that it is possible to set and get properties of packets, store or discard them, apply delays, make module specific operations.

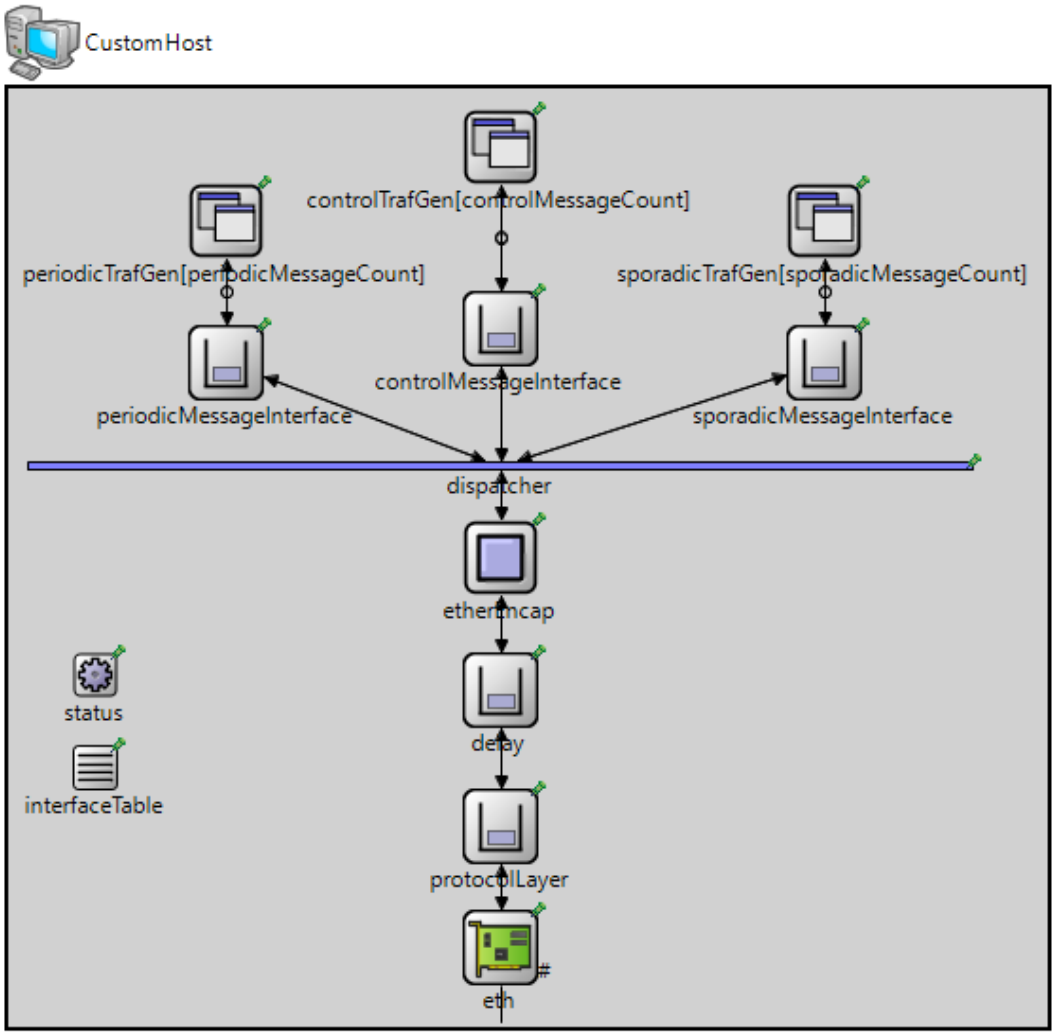

Figure 4.1: SQDSR Host OMNeT++ Model

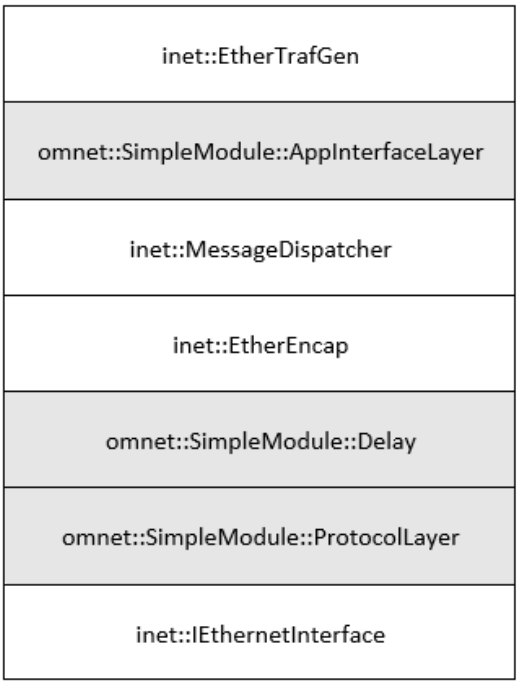

Figure 4.2: Types of Network Components in SQDSR Host Model

The designed model is configurable based on the parameters given in Table 4.1. Since the protocol is based on cycles; the periods of them should be parametric. There are some fixed size slots such as control slots and end of cycle slot which should also be configurable during design time. `Delay` component takes a parameter to decide on the amount of delay introduced in order to simulate latency caused from operating system, framing and deframing. Determining the synchronization and retransmission master nodes are also available during configuration process.

4.2.2.1 Traffic Generator and Interface Components

SQDSR protocol handles three types of packets; control packets, periodic packets and sporadic packets. A simple SQDSR node is able to generate each of them, but should generate at least one control packet for each high level cycle. To this end, there are three instances of `EtherTrafGen` that we call `controlTrafGen`, `periodicTrafGen` and `sporadicTrafGen`. These instances are stored in an array with a parametric length for flexible network configuration. The configuration

Parameters	Simulation Values
Node Count	23
Low Level Cycle Period	1 ms
High Level Cycle Period	5 ms
Control Slot Length	10 μs
End of Cycle Slot Length	10 μs
Processing Delay	15 μs
Bandwidth	100 Mbps

Table 4.1: Configurable SQDSR Simulation Parameters

of traffic generators are done by the parameters; offset, period, data length of message and the destination address.

The instances of `EtherTrafGen` are directly connected to `MessageDispatcher` in standard `EtherHost` model. Nevertheless, an additional layer is inserted between them with the purpose of assigning message IDs to generated packets and recording simulation statistics of packets. The layer is named based on the traffic generator it follows. Even if their names are different, these layers are completely identical. `AppInterfaceLayer` is created from OMNeT++ simple module. It is a transparent layer that has no effect on simulation process. It is used to keep the records of the periodic and sporadic packets including latency histogram, average/max latency for periodic and sporadic messages. The delays are measured end to end; from transmitter `AppInterfaceLayer` to receiver `AppInterfaceLayer`. This layer also assigns message IDs help tracking each distinct packet performance. A message ID is a combination of source node ID, defined index of the message in that node and the message type where it is 0x00 for control messages, 0x01 for periodic messages and 0x02 for sporadic messages. The format of message ID is available in Fig 4.3. The assigned message IDs for the whole message set can be seen in Table 4.6.

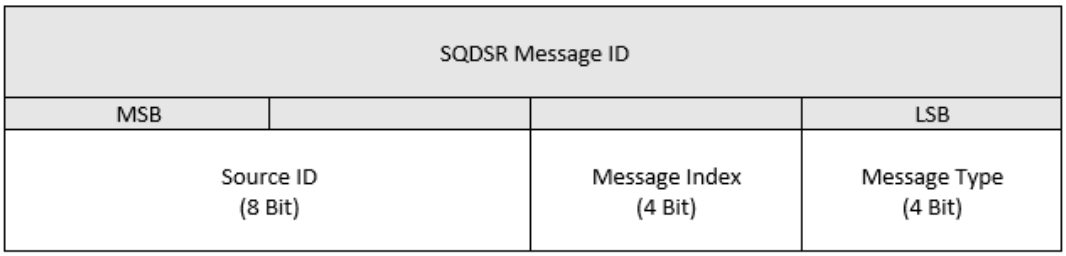

Figure 4.3: SQDSR Message ID

4.2.2.2 Delay Component

SQDSR model contains another custom component, `Delay`, simulating the processing delay due to operating system, framing, and deframing. Since the processing delay is mostly dependent on the hardware and operating system, there is no similar component designed for in OMNeT++ or INET framework. However, there is one in AFDX model described in 4.2.3. The model is smoothly portable and used in SQDSR model. The component requires the value of delay parameter.

The paper [22] states that a guard time of 15μs is a sufficient duration as TDMA slot length over a network transmitted Ethernet frames without any higher level protocols. The same paper states that it is difficult to achieve a network response time less than 10s of microseconds for an application level software. Furthermore, idle Linux kernel can transmit and receive a UDP packet within a round trip time of 32μs [20]. Considering these results, the `Delay` component is modeled to introduce 15μs for transmitted packets from application layer to protocol layer and for received packets from protocol layer to application layer.

4.2.2.3 SQDSR Protocol Component

The layer handles all functionality of SQDSR protocol. It is on top of the INET Ethernet MAC and physical layer. The outgoing sporadic and periodic packet requests are queued in this layer with a special container that orders packets based on their deadlines. The packet on top of the periodic queue is removed and sent in static slot and the packet on top of the sporadic queue is removed and sent in dynamic

slot assigned to the corresponding node. Control packets are generated and sent in assigned time slots. IEEE1588 time synchronization protocol is applied in this layer to synchronize internal clocks of nodes as described in Section 3.3.

The layer has three states of operation; control, static and dynamic as the details of the states explained in Section 3.2. State transitions are occurred with timer interrupt, packet reception or transmission. Control state starts and ends in predefined time points based on low level and high level cycle periods. Following the control state, static state starts and progresses based on the schedule predetermined. There are no time divided slots in static and dynamic states. The operations are triggered by reception of packets. Since each node is able to analyze packets and know the next schedule, there is no need for a time dependent mechanism.

The main responsibility of this layer is to store the shared queue that involves the packet records from different nodes. These records determine the next schedule in dynamic state of network based on the deadline information as priority. Unless a sporadic packet is notified and inserted to shared queue, it is not allowed to be transmitted in dynamic state. Retransmission of erroneous packets is also handled in this network layer. The error reports are received at the beginning of the retransmission cycle and the nodes are reacted accordingly the faults. To be able to retransmit, the copy of the previous packets should be stored internally. Since the retransmission cycle repeats in each low level cycle, it is sufficient to store only the packets sent in the last cycle. If the node is the retransmission master, then the fault report need to be generated for each previous cycle.

4.2.3 AFDX OMNeT++ Model

There is an AFDX model for OMNeT++ available under simulation models and tools [23]. The model was developed by Rudolf Hornig and includes basic AFDX MAC layer and AFDX switch implementation. The model offers redundancy management and queuing in both switches and end systems. The available real time traffic set should be applied to this model in order to compare it with SQDSDR protocol. The architecture of given network consists of 2 switches and 23 nodes. The Fig. 4.4 shows the representation of the given network in OMNeT++ AFDX model.

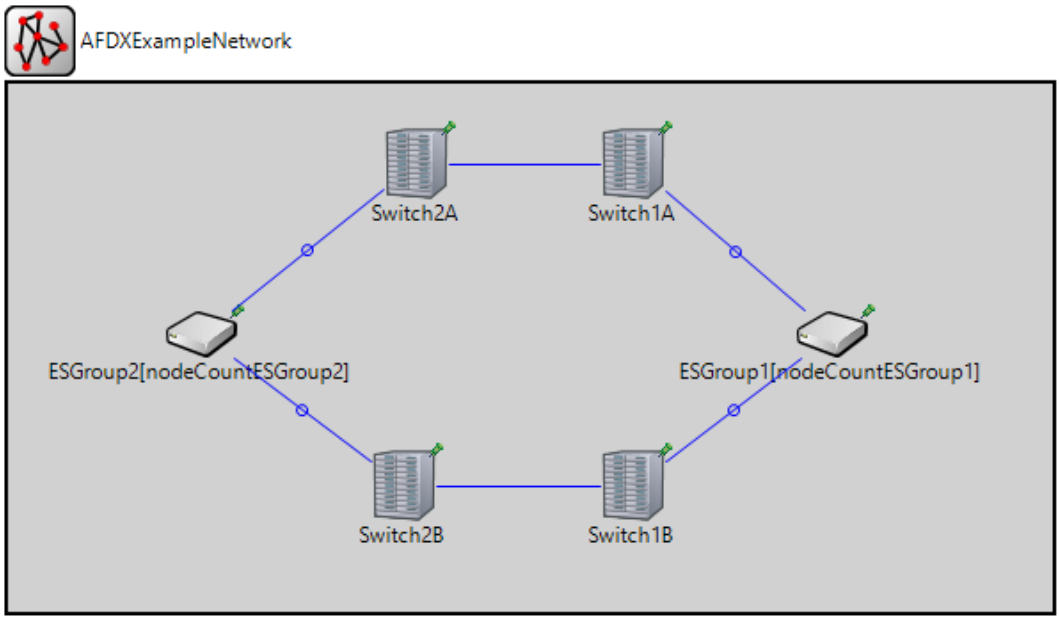

Figure 4.4: Simulated AFDX OMNeT++ Network with Two End System Groups and Redundant Switches

`ESGroup1` and `ESGroup2` refer to end systems connected to either `Switch2` or `Switch1`. They are declared as arrays of nodes with configurable size; `nodeCountESGroup1` and `nodeCountESGroup2`, which are equal to 10 and 13 respectively for the given traffic. As seen in Figure 4.4, there are two instances of each switches as A and B. The reason is providing hardware redundancy on switches by sending duplicate messages from end systems to the both switches A and B.

4.2.3.1 AFDX End System Model

AFDX `EndSystem` model is demonstrated in Fig. 4.5. The original AFDX model does not contain `LatencyTransmission` and `LatencyReception` components, so they were added to represent the delay due to AFDX end system technology. It is by definition the time required to accept, process and begin transmission of frame when the system is idle [6]. They do not refer to the latency due to contention or queuing. Furthermore, AFDX ES model has `RegulatorLogic` unit but does not have the BAG control implementation as discussed in Chapter 2.3.4. The

regulator algorithm is realized in Regulator Logic unit based on the specifications [6]. Another modification has been made in the components `TrafficSource` and `TrafficSink` to gather statistics of sent and received packets including the average and maximum latency. In other words, the packet statistics are always end to end; from ES where packet is generated to the ES where it is received. Note that all the modifications comply with the original simulation model and specifications [6].

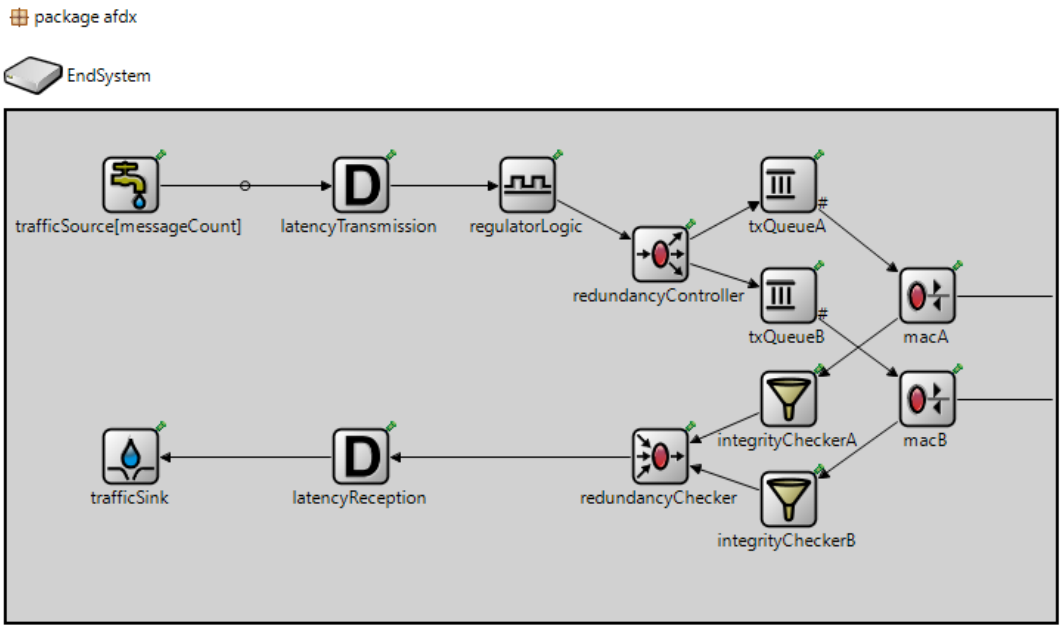

Figure 4.5: AFDX End System OMNeT++ Model

The configuration parameters of AFDX `EndSystem` model is available in Table 4.2. The network traffic is regulated with respect to BAG and jitter values. BAG is set based on the assigned VL IDs of messages in Table 4.3. BAG value should be at least 1 and power of 2, logically should be less than or equal to the period of message [6]. Latency is configured as 50μs according to specifications from the manufacturer of end system. The other parameters including minimum and maximum jitter values are kept with default values in model for the simulation. AFDX protocol is based on UDP by design, packet overhead parameter is defined as 47 bytes to include overhead of Ethernet, IP and UDP frames.

Parameters	**Simulation Values**
Minimum Jitter	40 μs
Maximum Jitter	500 μs
Latency Transmission	50 μs
Latency Reception	50 μs
TX Queue Capacity	Infinite
Queue Algorithm	Priority
MAC Fetching Algorithm	Round-robin
Packet Overheads	47 Bytes
Supported Bit Rate	100 Mbps

Table 4.2: Configurable AFDX Simulation Parameters

BAG	**VL IDs**
1 ms	0x2000, 0x2100, 0x2A00, 0x2A01, 0x2A02 0x2B00, 0x2B01, 0x2B02, 0x1900
4 ms	0x2400, 0x2500, 0x2600, 0x2700, 0x2C00 0x2C01, 0x2C02, 0x2C03, 0x2C04, 0x2C05 0x2C06, 0x2C07, 0x2C08, 0x2C09, 0x2C0A 0x2C0B, 0x1000, 0x1100, 0x1200, 0x1300 0x1400, 0x1500, 0x1600, 0x1700, 0x1800
32 ms	0x2200, 0x2300
64 ms	0x2800, 0x2C0C

Table 4.3: Configured AFDX Bandwidth Allocation Gap (BAG) Values based on VL IDs from Table 4.6

4.2.3.2 AFDX Switch Model

AFDX switch model consists of a switch fabric and multiple switch ports. The modules of fabric and port are shown in Fig. 4.6. `SwitchPort` contains `MAC` to receive and send frames; `FrameFilter` to verify virtual link identity (VL ID) incoming port pairs and the availability of destination port; `TrafficPolicy` to check the defined bandwidth limit for VL ID and accept incoming packets based on token bucket algorithm; `TxQueue` to convey packets to MAC and store the bursts. `SwitchPort` consists of `Classifier` to direct packets to correct priority queue; `Scheduler` to select packets from priority queues; `Delay` to simulate total processing time consumed on switch; `Router` to route packets to correct ports based on VL IDs.

AFDX switch model requires to have an implementation for VL routing in order to forward packets to the correct ports. The solution is providing routing tables to each switches that includes mappings for VL IDs to destination ports. Routing files differ from one switch to another since they expect different VL IDs. Once a packet received, destination ports are resolved and the packet is forwarded to corresponding ports. If the packet's VL ID is not in the configuration table, it will be dropped. In the scope of thesis, we have two switches, therefore, two routing table. The VL ID to destination ports mapping is given in routing Table 4.4 and Table 4.5 where VL IDs are assigned to the message set in Table 4.6. The tables are created as text files, parsed and distributed to switches at run time. The port number 23 is determined as the port destined to the other switch for both switches. If the destination and source nodes are connected to different switches, the packet is forwarded to port 23.

AFDX switches have traffic policing capability based on token bucket explained in Section 2.3.4 where packets are dropped if there is no adequate credit for that VL ID. The credit increases as time passes with rate (2.3.1). It is set to the maximum possible credit (2.3.2) at startup. Packets having insufficient credit are dropped in switch. The token bucket algorithm is not implemented in the original model that's why it is developed regarding to the specifications [6].

AFDX switch model is designed to process frames within 50μs with respect to the given specification from the manufacturer. It is configured with `Delay` component of

VL ID	**Destination Ports**
0x2000, 0x2100, 0x2200, 0x2300 0x2500, 0x2600, 0x2700, 0x2800	(4, 9, 12)
0x2400, 0x2A00, 0x2A01, 0x2A02 0x2B00, 0x2B01, 0x2B02	(9)
0x2C00	(4, 5, 9)
0x2C01	(4, 6, 9)
0x2C02	(4, 7, 9)
0x2C03, 0x2C04, 0x2C05, 0x2C06 0x2C07, 0x2C08, 0x2C09, 0x2C0A 0x2C0B	(4, 9, 23)
0x2C0C	(4, 8, 9)

Table 4.4: AFDX Routing Table for Switch 2

VL ID	**Destination Ports**
0x1000, 0x1100, 0x1200, 0x1300 0x1400, 0x1500, 0x1600, 0x1700 0x1800, 0x1900	(23)
0x2C03	(0)
0x2C04	(1)
0x2C05	(2)
0x2C06	(3)
0x2C07	(4)
0x2C08	(5)
0x2C09	(6)
0x2C0A	(7)
0x2C0B	(8)

Table 4.5: AFDX Routing Table for Switch 1

AFDX switch fabric model shown in Fig. 4.6. This amount of delayed is introduced to packets passing through each switch. Therefore, the packets crossing multiple switches experience more delays. Apart from delay, AFDX switch parameters have similar with end system ones including queuing properties, fetching algorithms and BAG.

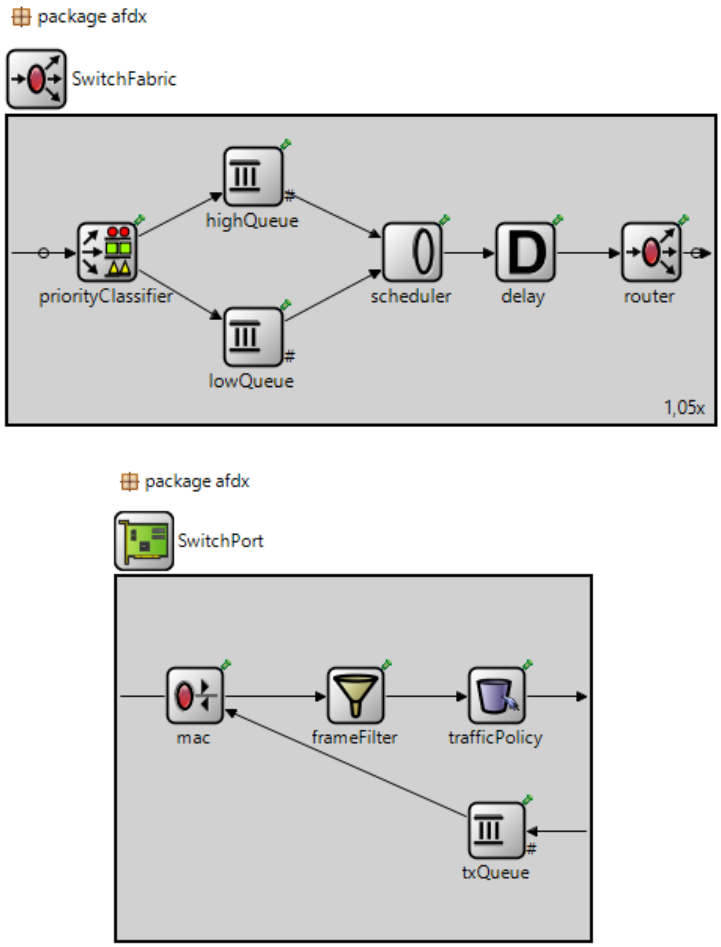

Figure 4.6: AFDX Switch Model Containing Switch Port and Fabric

4.3 Experimental Results

The simulation models in previous chapters were tested under real network architecture and message set. It was supplied by one of the commercial company at avionics industry. The network consists of 23 nodes where most of the messages are periodic.

In the scope of the thesis, the proposed protocol SQDSR is compared with AFDX protocol with the given message set, shown in Table 4.6. The network consists of two switches, where 13 nodes are connected to Switch 2 and 10 nodes are connected to Switch 1. The switched configuration are applied directly to AFDX model. However, it is not possible to use it with SQDSR simulation since it is a shared bus based protocol without any switches. Therefore, switches are removed and all nodes in the network are connected to the same shared bus.

If the message set in Table 4.6 is examined, the data length of packets is lower than maximum Ethernet frame data length, 1500, except from the last message generated by node 0x16. It has 5000 bytes data per frame so it should be fragmented. The message is partitioned into 4 packets with the length of 1471, 1471, 1471 and 587 bytes respectively due to the maximum data length carried on AFDX payload. They share the same VL ID whereas they have different SQDSR message IDs; 0x1601, 0x1611, 0x1621 and 0x1631 based on the message ID assignment method in Fig. 4.3 and referred as 0x16X1 in Table 4.6. It is seen that most of the messages are periodic. There are also sporadic messages and two of them have length of 1316 bytes with periods of 421 μs. These are cameras producing H.264 encoded videos. According to [25], the streams with H.264 encoding are sliced into 188 bytes called Packetized Elementary Stream (PES). It is better to combine PES packets into single frame to eliminate frame overheads. Therefore, single Ethernet frame is able to carry 7 of them which is equal to 1316 bytes data length. The specifications indicate that the camera nodes can operate at 25 Mbps rate; hence the period of camera data with 1316 bytes data length should be 421 μs. However, minimum value of BAG is defined as 1ms in AFDX specification which results in blocking all packets having smaller periods at traffic regulation. Since this is also valid for camera data, it is not possible to carry it in a single VL ID. The problem is managed by partitioning the data into three VL IDs such that their periods become 1.263 ms which is no longer violating our BAG condition. For camera node 0x0A, the corresponding VL IDs are 0x2A00, 0x2A01 and 0x2A02 referred as 0x2A0X; whereas they are 0x2B00, 0x2B01 and 0x2B02 referred as 0x2B0X in Table 4.6 for camera node 0x0B.

Source ID	Switch Port	Message Period	Data Length	Message Type	SQDSR ID	AFDX VL ID
0x00	SW2 - P0	1 ms	250 Bytes	Periodic	0x0001	0x2000
0x01	SW2 - P1	1 ms	750 Bytes	Periodic	0x0101	0x2100
0x02	SW2 - P2	50 ms	750 Bytes	Periodic	0x0201	0x2200
0x03	SW2 - P3	50 ms	750 Bytes	Periodic	0x0301	0x2300
0x04	SW2 - P4	5 ms	100 Bytes	Sporadic	0x0402	0x2400
0x05	SW2 - P5	5 ms	200 Bytes	Periodic	0x0501	0x2500
0x06	SW2 - P6	5 ms	200 Bytes	Periodic	0x0601	0x2600
0x07	SW2 - P7	5 ms	100 Bytes	Periodic	0x0701	0x2700
0x08	SW2 - P8	100 ms	1000 Bytes	Periodic	0x0801	0x2800
0x0A	SW2 - P10	421 μs	1316 Bytes	Sporadic	0x0A02	0x2A0X
0x0B	SW2 - P11	421 μs	1316 Bytes	Sporadic	0x0B02	0x2B0X
0x0C	SW2 - P12	5 ms	200 Bytes	Periodic	0x0C01	0x2C00
0x0C	SW2 - P12	5 ms	200 Bytes	Periodic	0x0C11	0x2C01
0x0C	SW2 - P12	5 ms	100 Bytes	Periodic	0x0C21	0x2C02
0x0C	SW2 - P12	5 ms	100 Bytes	Periodic	0x0C31	0x2C03
0x0C	SW2 - P12	5 ms	200 Bytes	Periodic	0x0C41	0x2C04
0x0C	SW2 - P12	5 ms	100 Bytes	Periodic	0x0C51	0x2C05
0x0C	SW2 - P12	5 ms	200 Bytes	Periodic	0x0C61	0x2C06
0x0C	SW2 - P12	5 ms	200 Bytes	Periodic	0x0C71	0x2C07
0x0C	SW2 - P12	5 ms	200 Bytes	Periodic	0x0C81	0x2C08
0x0C	SW2 - P12	5 ms	100 Bytes	Periodic	0x0C91	0x2C09
0x0C	SW2 - P12	5 ms	100 Bytes	Periodic	0x0CA1	0x2C0A
0x0C	SW2 - P12	5 ms	100 Bytes	Periodic	0x0CB1	0x2C0B
0x0C	SW2 - P12	100 ms	1000 Bytes	Periodic	0x0CC1	0x2C0C
0x0D	SW1 - P0	5 ms	100 Bytes	Periodic	0x0D01	0x1000
0x0E	SW1 - P1	5 ms	200 Bytes	Periodic	0x0E01	0x1100
0x0F	SW1 - P2	5 ms	100 Bytes	Periodic	0x0F01	0x1200
0x10	SW1 - P3	5 ms	200 Bytes	Periodic	0x1001	0x1300
0x11	SW1 - P4	5 ms	200 Bytes	Periodic	0x1101	0x1400
0x12	SW1 - P5	5 ms	200 Bytes	Periodic	0x1201	0x1500
0x13	SW1 - P6	5 ms	100 Bytes	Periodic	0x1301	0x1600
0x14	SW1 - P7	5 ms	100 Bytes	Periodic	0x1401	0x1700
0x15	SW1 - P8	5 ms	100 Bytes	Periodic	0x1501	0x1800
0x16	SW1 - P9	5 ms	5000 Bytes	Periodic	0x16X1	0x1900

Table 4.6: Properties of Simulated Real Case Message Set

4.3.1 SQDSR Simulation Results

SQDSR OMNeT++ simulation is configured with the parameters in Table 4.1 and scheduled as in Fig. 4.7. It is assumed that the reaction time between receiving a packet and forwarding the queued one is negligible during the simulation. The simulated network is highly loaded such that the utilization is measured as % 79 by means of the reception status of any node thanks to the broadcasted architecture. End to end delay is measured for each packet during the simulation and the overall results per message ID are available in Fig. 4.10. The latency measurements are obtained by finding the difference between the time of the last bit of packet received at the receiver node and the time packet generated at the transmitter node.

Since the protocol has static slots reserved for periodic messages, it is necessary to schedule the simulated message set in Table 4.6. There is an offset column describing the low level cycle that the message is generated. The offsets are adjusted such that the schedule is appropriate to handle all the messages assigned to that low level cycle. Since the high level cycle period is 5 ms, the range of offsets changes between 0s to 5ms. The sample schedule used in simulation is shown in Fig. 4.7. The first cycle is deliberately less loaded due to the existing control slots at the beginning of it.

	Each 1 ms		*Each 5 ms*								*Each 50 ms*	*Each 100 ms*
Cycle 0	0x0001	0x0101	0x0501	0x0C21	0x0C71	0x0D01	0x1201					
Cycle 1	0x0001	0x0101	0x0601	0x0C31	0x0C81	0x0E01	0x1301				0x0201	
Cycle 2	0x0001	0x0101	0x0701	0x0C41	0x0C91	0x0F01	0x1401				0x0301	0x0801
Cycle 3	0x0001	0x0101	0x0C01	0x0C51	0x0CA1	0x1001	0x1501					0x0CC1
Cycle 4	0x0001	0x0101	0x0C11	0x0C61	0x0CB1	0x1101	0x1601	0x1611	0x1621	0x1631		

Figure 4.7: Schedule of Message Set for SQDSR Simulation

Fig. 4.8 demonstrates the effects of schedule on three different messages; 0x0001,

0x0101 and 0x1201. It shows end to end latency results based on the number of packets per message ID. The figure is important to understand the fundamentals of protocol static state explained in Section 3.2.2. The messages 0x001 and 0x0101 have the same period of 1 ms and scheduled successively at the beginning of the cycle as indicated in schedule Fig. 4.7. However, `Cycle-0` is special for having control slots for each node, which increases the queuing time of packets, therefore, end to end latency, assuming that the packets are generated at the beginning of cycles. Since `Cycle-0` is proportionally 20 % of the overall high level cycle, the same proportion is valid for the latency results of packets with message IDs 0x001 and 0x0101. The simulation runs for 10 seconds, so the number of packets generated is 10.000 for both of them. On the other hand, the packet with message ID 0x1201 has 5 ms period so it produces 2000 packets at the end of the simulation. It is again scheduled at `Cycle-0` according to Fig. 4.7. However, it does not have distinct latency values as the previous messages, since it has solely a single slot over the whole high level cycle.

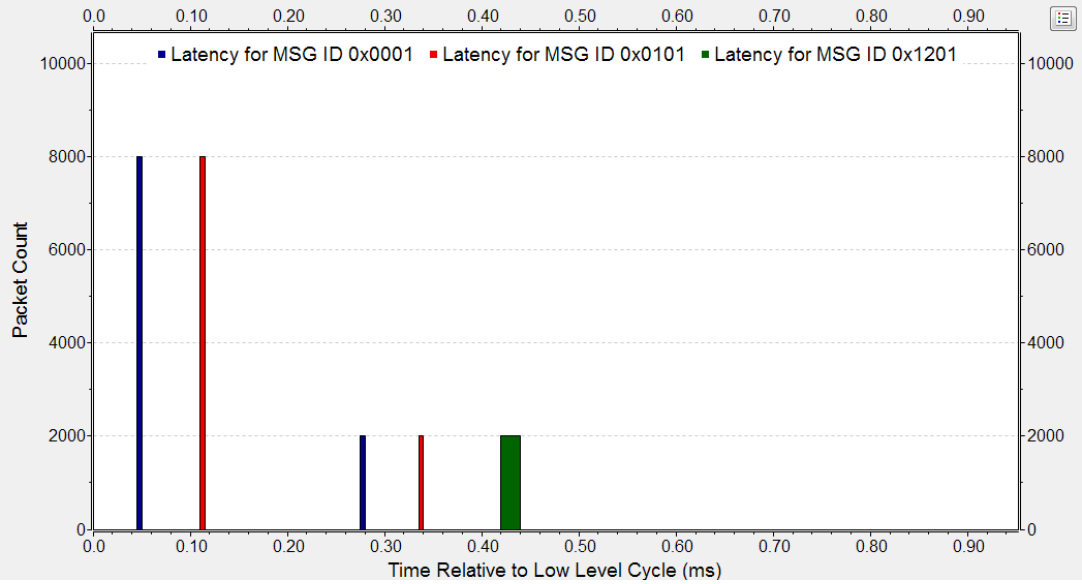

Figure 4.8: Observing The Effects of SQDSR Control Slots and Periodic Schedule over Three Different Messages

Reception Status	Percentage
Rx Idle	% 21
Rx Active	% 79

Table 4.7: Measuring Network Utilization of Simulated SQDSR based on Rx Status of Nodes

Message Type	Avg. End to End Latency	Average Queuing Delay	Max. End to End Latency	Maximum Queuing Delay
Periodic	0.214 ms	0.170 ms	0.595 ms	0.533 ms
Sporadic	1.971 ms	1.854 ms	5.484 ms	5.364 ms

Table 4.8: SQDSR Queuing Delay and End-to-End Latency Results for Periodic and Sporadic Traffic Types

4.3.1.1 SQDSR Performance

The performance of SQDSR are observed with simulation of 10 seconds and available in Table 4.8. It is assumed that nodes are able to react instantly so there is no processing time between static slots or dynamic slots. The results show that periodic messages have 0.214 ms average latency and 0.595 ms maximum latency. There is an improvement in terms of the latency for periodic messages compared to AFDX simulation results in Table 4.13. Even though there is a small deterioration for average end to end latency of sporadic messages, it is seen that SQDSR outperforms than AFDX in terms of maximum end to end latency. The lack of additional delays due to switches and not having traffic regulation have significant role in this enhancement. The Table 4.8 also contains average and maximum queuing delays packets experience. The queuing delay is the component of the latency measurement excluding the propagation and processing delays.

The reason behind the poor performance of sporadic packets is that the nodes transmitting them do not have any periodic messages, therefore no static slots. For this

Message Type	Avg. End to End Latency	Max. End to End Latency
Periodic	0.215 ms	0.615 ms
Sporadic	1.037 ms	3.061 ms

Table 4.9: SQDSR Average and Maximum End-to-End Latency Results After Modification to Message Set

reason, control slots are the only way to notify the sporadic messages which repeat only in high level cycles within 5 ms intervals. Hence, the latency of the sporadic messages rises up to the period of high level cycle. To overcome the disadvantageous situation for sporadic packets, the message set is modified. The modification keeps all the message set as it is, but involves additional one periodic message per node who transmit sporadic messages; i.e. the nodes 0x04, 0x0A and 0x0B in Table 4.6. The add-on periodic messages have 1 ms periods with minimum Ethernet frame length. They are served as additional notification points for the new sporadic message requests. The results for the modified message sets are available in Table 4.9 indicating the latency of sporadic messages is reduced roughly by %50, now better than the average latency result of AFDX.

The effect of having periodic slots is also measured in an another simulation with original message set but one of the camera node has reserved periodic slot in each cycle. The sporadic queue lengths of two camera nodes are compared and the cumulative distribution function (CDF) is deducted as a result, plotted in Fig. 4.9. The upper line indicates the camera nodes having the extra periodic slots, whereas the other does not have any. The results justify that average length of the sporadic queue is reduced and the maximum length of it goes down to 14 from 18, if the node producing sporadic messages also has periodic slots. Again, the effects of periodic slots as an additional notification point to update shared queue is observed.

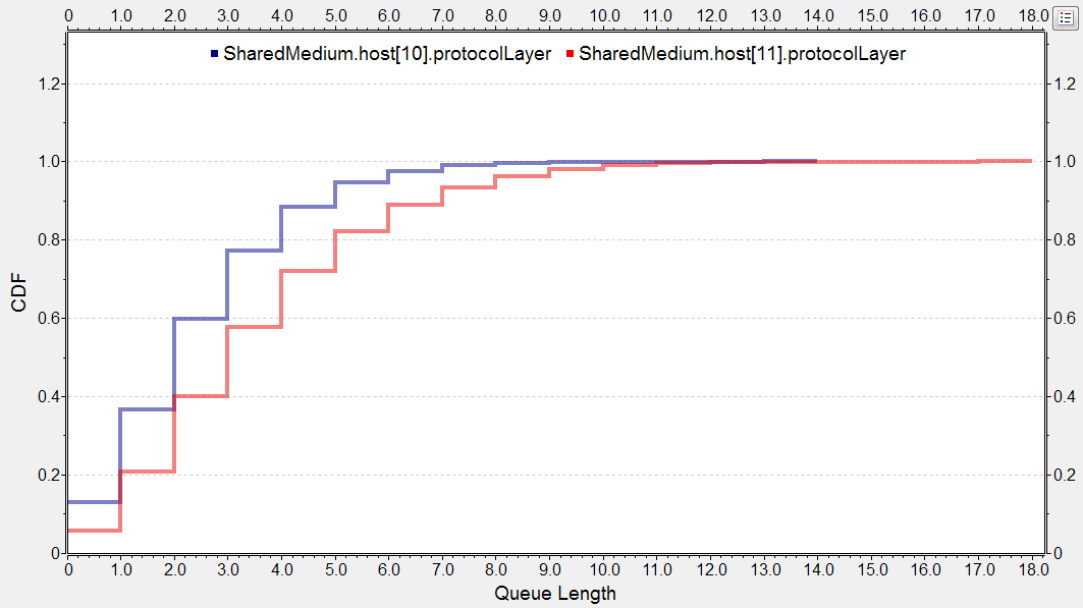

Figure 4.9: SQDSR Sporadic Queue Length Cumulative Distribution Function for Two Camera Nodes where the Upper Line Indicating the One with Additional Periodic Slots

The measurements for each message ID are also obtained to compare them with equivalent AFDX messages. The results conclude that messages destined to other switched network in AFDX model performs quite worse than the SQDSR model. Periodic message performances differ depending on the schedule order in the cycle. Even if the simulation is realized in a highly loaded network, the queues are not growing during simulation; plus, there is no message missing its deadline. Hence, the network is schedulable for SQDSR protocol architecture. The latency figures would be better in low traffic network conditions for sporadic messages. The simulation results point that SQDSR performs sufficiently and could be a good alternative for AFDX protocol.

Figure 4.10: SQDSR Simulation Latency (ms) Measurements per Message ID

4.3.1.2 SQDSR Fault Handling Observation

SQDSR is designed to handle any faults occurred thanks to the broadcasted communication. The retransmission master listens the network consistently to detect any inconvenient situation. As discussed in Section 3.5, SQDSR retransmission notification frame is transmitted at the beginning of dynamic state in order to convey the faulty packets information to the nodes. It is expected that the nodes responsible for the faults should retransmit the packets based on the same order in the notification frame. A simple simulation is performed to illustrate the retransmission procedure with mes-

sage IDs 0x0501 and 0x1201. Note that both of them are scheduled at `Cycle-0` as seen in Fig. 4.7 whereas the latter is assigned to the last periodic slot in that cycle. They both have 5 ms periods, hence, producing 2000 packets at the end of 10 seconds simulation. It is expected to have retransmission notification frame just after the periodic slot reserved for 0x1201 and the retransmission of erroneous packets follow it.

The frame with message ID 0x0501 and 0x1201 are transmitted with probability of 50 % and 20 % fault respectively. The left-most two lines in Fig. 4.11 represents the reception time of non-faulty packets relative to the low level cycle for message IDs 0x0501 and 0x1201 respectively. The number of non-faulty packets matches with the fault probabilities. If one of these packets fails, retransmission is realized and the latency values around 0.45 ms are obtained. However, it is also possible to both packets could fail at the same cycle. It ends up with more delay for 0x1201 since it is going to be located at the second position on retransmission notification frame. Hence, it should wait till the the first faulty frame 0x0501 completed its retransmission. It is assumed that the retransmitted frames are completed without any fault.

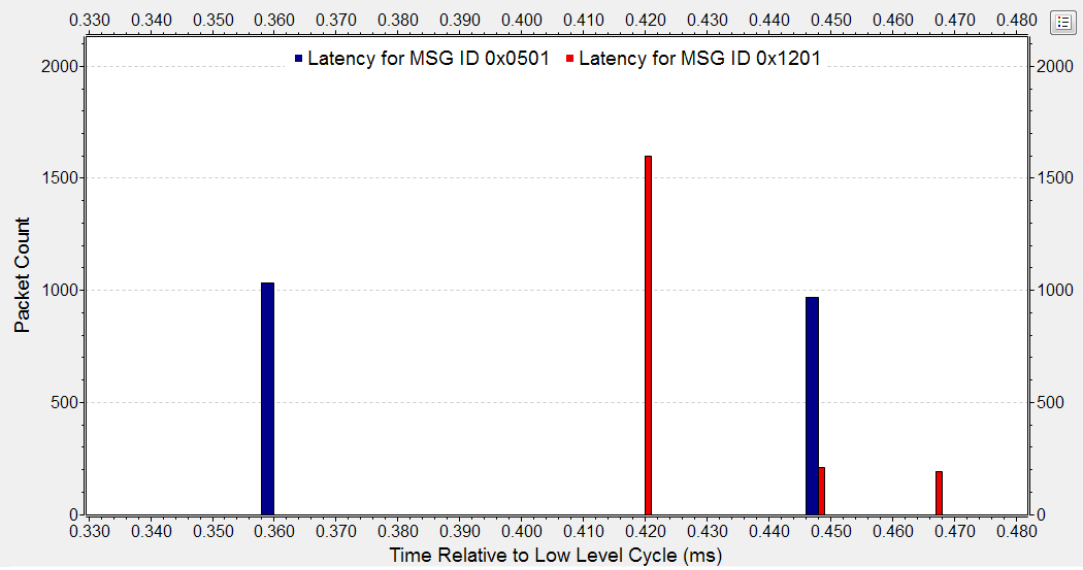

Figure 4.11: Observing The Effects of Faults on SQDSR Latency Results

4.3.1.3 Effects of Startup Configuration on SQDSR Results

The previous simulations for SQDSR are performed under assumption that periodic packets are generated at the beginning of the cycle where they have the scheduled static slots. Though this is the ideal case for SQDSR, it is not an obligation. The packets could be generated at any time and served at their scheduled slots. This apparently induces more queuing time which is bounded by the period of packet. Therefore, it is important to test the proposed protocol response time under different startup configurations apart from the default case (synchronized with offsets).

First, it is assumed that there is no synchronization at all for both inter-nodes and intra-node during packet generation. In other words, every periodic packet has distinct start time assigned randomly. The results in Table 4.10 show that the maximum queuing delay goes up to 100 ms as expected which is the largest period in our message set in Table 4.6 that could also be observed in schedule table Fig. 4.7 (messages with IDs 0x0801 and 0x0CC1). Average queuing delay also increases compared to the default configuration since the packets are not generated in the cycle where they have scheduled static slots. The same observation is valid for the next simulation configuration where the nodes are still not synchronized but each node is able to generate their packets synchronously. However, the possibility of missing the first assigned slot still exists which ends up with additional delay up to the periods of packet. The last configuration involves synchronous nodes producing packets at t_0. It is seen that the worst case response time of periodic messages is bounded by the period of high level cycle which is determined as 5 ms for the simulation. In this configuration, all messages have the chance to transmit at their first scheduled slots, so the performance is improved for periodic packets. It is remarkable that sporadic messages are not affected significantly from different configurations since they are always generated randomly independent from the startup of nodes and they could be served at the first dynamic slot available.

Startup Configuration	Avg. Queuing Delay	Max. Queuing Delay
No Synchronization	Periodic: 2.476 ms Sporadic: 1.898 ms	Periodic: 100 ms Sporadic: 5.602 ms
Synchronization In-Node	Periodic: 2.767 ms Sporadic 1.968 ms	Periodic 100 ms Sporadic 5.366 ms
All Synchronized at t_0	Periodic 1.803 ms Sporadic 1.868 ms	Periodic 4.532 ms Sporadic 5.363 ms
Synchronized with Offsets (Default Configuration)	Periodic 0.170 ms Sporadic 1.854 ms	Periodic 0.533 ms Sporadic 5.364 ms

Table 4.10: SQDSR Queuing Delay Results Under Different Start Time Configuration of Nodes

4.3.1.4 Effects of Network Configuration on SQDSR Results

The simulated network consists of 23 nodes producing both sporadic and periodic messages. The low level cycle period is selected as 1 ms based on the greatest common divisor of message periods. The high level cycle period of SQDSR is selected as 5 ms with regard to the traffic load and message periods. It is noteworthy to demonstrate the effects of selected network parameters on SQDSR simulation results. To this end, the section covers the results of additional simulations with different network configurations.

First, the impact of high level cycle period needs to be investigated. As it determines the frequency of control state, it provides notification point for shared queue requests. However, there is an excessive wasted bandwidth utilization once they exist in a cycle. If the network is highly loaded, the less high level cycle period could result in smaller space left for dynamic state, i.e. more queuing time for sporadic messages. Hence, it is essential to select the most appropriate value. The simulation result in

Message Type	Average Latency	Maximum Latency
Periodic	0.248 ms	0.823 ms
Sporadic	3.128 ms	12.200 ms

Table 4.11: Average and Maximum Latency Results of SQDSR Simulation with Reduced High Level Cycle Period (3 ms)

Table 4.11 shows the effects of smaller high level cycle period by selecting it as 3 ms. The sporadic packets are not the only ones affected, but also periodic packets perform worse due to the extra offset for control slots at the beginning of every three cycles. Incidentally, this period is the lowest possible integer value that the network is still schedulable. Lower figures result in constantly growing queues because of the sporadic packets and the network would not be stable anymore. However, more frequent high level cycle does not always mean poorer performance, which will be discussed in the following simulation. It depends on the network traffic load. It could enhance dynamic slot allocation process with more frequent control slots and therefore producing better sporadic performance if the network has low traffic.

Next, it is significant to demonstrate the effects of node count on SQDSR performance. To this end, number of nodes is increased to 32, which is the maximum allowed number of nodes for SQDSR by design. As the number of nodes increases, the length of control state extends which causes worse performance on latency results. Nevertheless, it is subtle to see the impact with 5 ms high level cycle period. Therefore, it is reduced to to 1 ms, now equals to the low level cycle period enabling to have control slots in every cycle. However, the original network with message set in Table 4.6 is too loaded for this configuration. For this reason, the network is modified such that the nodes apart from node ID 0x00, 0x01, 0x0A and 0x0C do not transmit anything. First, the custom network is simulated with default configuration; 5 ms high level cycle period and 23 nodes. Then, the period is reduced to 1 ms while conserving the same number of nodes. Finally, it is increased to 32 in order to demonstrate the impact of node count.

Table 4.12 indicates that reducing high level cycle period might improve sporadic

Network Configuration	Avg. Queuing Delay	Max. Queuing Delay
High Level Cycle Period: 5 ms Node Count: 23	Periodic: 0.114 ms Sporadic: 1.834 ms	Periodic: 0.339 ms Sporadic: 4.879 ms
High Level Cycle Period: 1 ms Node Count: 23	Periodic: 0.301 ms Sporadic 0.578 ms	Periodic 0.442 ms Sporadic 1.912 ms
High Level Cycle Period: 1 ms Node Count: 32	Periodic 0.391 ms Sporadic 0.635 ms	Periodic 0.532 ms Sporadic 2.232 ms

Table 4.12: The Observations on Custom SQDSR Network with Active Nodes 0x00, 0x01, 0x0A and 0x0C

packet's performances. It is the consequence of having more chance to notify lately generated sporadic packets by means of control slots in order to make slot reservation. The impact is opposite to the results in Table 4.11 as the network is more idle for this simulation. Furthermore, the table shows that increasing the node count affects negative even if the add-on nodes do not produce packets to transmit.

4.3.2 AFDX Simulation Results

AFDX OMNeT++ simulation was performed with the message set in Table 4.6 as we did for SQDSR simulation. The network architecture consisting of 2 switches and 23 nodes was modelled with AFDX OMNeT++ model. The model is configured with the parameters in Table 4.2. The routing tables per VL IDs were created as detailed in the previous Chapter 4.2.3. The simulation of AFDX model with the specified message set is performed for 10 seconds under 100Mbps line speed. The packets are generated randomly based on their fixed periods and data lengths. End to end delay is measured for each packet during the simulation. Latency measurements are obtained by finding the difference between the time of the last bit of packet received at the receiver node and the time packet generated at the transmitter node.

Message Type	Average Latency	Maximum Latency
Periodic	0.335 ms	3.720 ms
Sporadic	1.210 ms	24.118 ms

Table 4.13: Average and Maximum Latency Results of AFDX OMNeT++ Simulation

4.3.2.1 AFDX Performance

The statistics of latency figures per virtual link ID are collected and summarized in Table 4.13. It shows that periodic messages have 0.335 ms average latency and 3.720 ms maximum latency, whereas the sporadic messages have 1.210 ms average latency and 21.118 ms maximum latency. The same message set performs worse than SQDSR simulation due to the additional delays and queuing occurred in switches and traffic regulation with BAG. The reason behind the maximum latency value of periodic messages is the regulated frame with 0x1900 VL ID. It consists of four successive packets which are blocked by the traffic regulation at ES and sent within 1 ms intervals that is the assigned BAG value for this VL ID. Even if sporadic messages perform better in terms of average latency, there could be excessive maximum latency up to 24 ms because of the traffic shaping mechanism when there is a burst of them.

The results contain 50 μs processing delays for each end system and switch the node passing through. The latency values should be examined with considering the packet sizes listed in message set Table 4.6. The latency figures for each VL ID are demonstrated in Fig. 4.12. Note that the camera data produced from node 0x0A and 0x0B are partitioned into three VL IDs due to the BAG traffic regulation, the corresponding VL IDs; 0x2A00, 0x2A01, 0x2A02 and 0x2B00, 0x2B01, 0x2B02 are reassembled to IDs 0x2A03 and 0x2B03 in simulation results respectively. This enables to gather statistics for each camera node so that it is comparable with SQDSR protocol message IDs 0x0A02 and 0x0B02.

Figure 4.12: AFDX Latency (ms) Results per VL ID

If the routing Table 4.4 and 4.5 are examined carefully, it will be observed that every packet is forwarded to Switch-2 Port-9 eventually. For this reason, the status of port's TX queue is investigated. To this end, the length of queue over time is obtained in Fig. 4.14. It indicates that number of packets waiting in the queue can reach up to 9 whereas the average is less than 2. The growing queue causes significant increase on packet's latency values. Apart from the queue length, the average queuing time spent in this queue is analyzed in Fig. 4.13. It states that on average 100 μs delay is introduced to packets and could be increased up to 600 μs. It is seen that jitter is

significant due to the constantly changing conditions in switch queues.

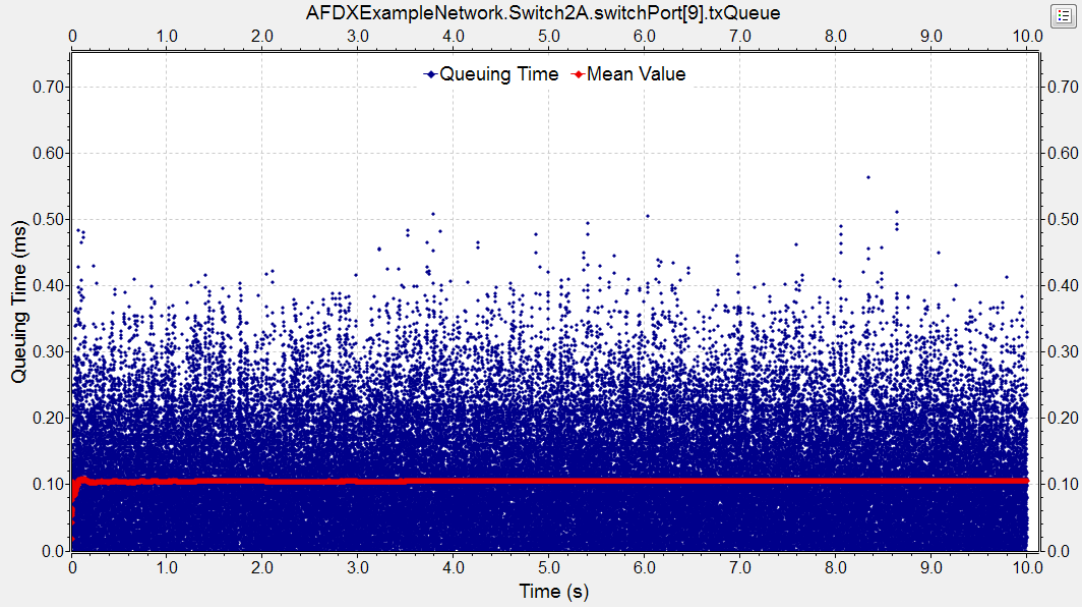

Figure 4.13: Plotted Time-varying TX Queuing Delay of AFDX Switch-2 Port-9

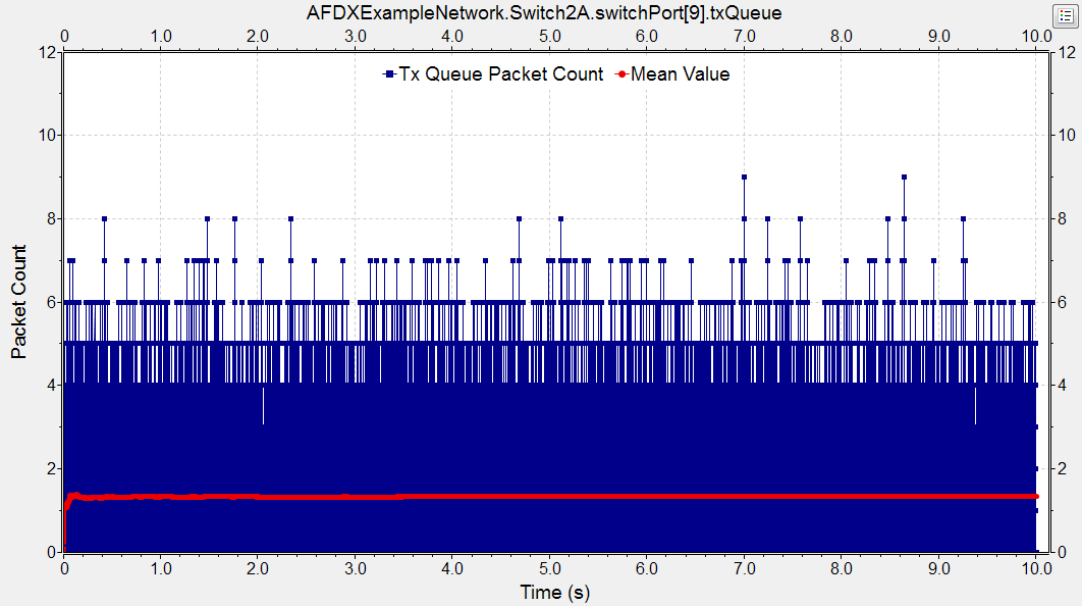

Figure 4.14: Plotted Time-varying TX Queue Length of AFDX Switch-2 Port-9

4.3.2.2 AFDX Fault Handling Investigation

Apart from the performance of AFDX, the fault handling mechanisms were also investigated. Note that the network shown in Fig. 4.4 contains redundant switches. Therefore, the transmitter ES forwards duplicate packets to the switches with same sequence number. Incidentally, every AFDX frame contains one byte sequence number field to discriminate redundant or invalid frames [6]. Thus, the receiver ES accepts the first packet arrived and discards the redundant one due to the repeated sequence number. The sequence number is kept per VL ID and incremented by one after each packet. Therefore, AFDX has tolerancy to faults occurred at switches, but it is not possible to handle faults happened at end systems. Dropped or corrupted packets in ES result in loss since AFDX has no retransmission mechanism.

The simulation to demonstrate the redundancy on AFDX is performed for the camera node whose data is partitioned into three VL IDs; 0x2A00, 0x2A01 and 0x2A02. At the end of 10 seconds simulation, each of them is expected to generate 7917 packets based on their periods given in Table 4.6. The number of packets received from main channel, i.e. not the redundant line coming from the redundant switch, and total number of packets received, i.e. from both channels, are measured respectively. The model is updated such that the packets with VL ID 0x2A01 is dropped with the probability of 20 % at main switch; while the packets with VL ID 0x2A02 is corrupted with the probability of 20 % at end system. The packets from 0x2A00 are left as they are for comparison. Fig. 4.15 justifies that the packets received from redundant channel are successfully discarded for 0x2A00 as the number of packets received from main channel and total number of packets are equal to the expected number 7917. Next, ES accepts some packets forwarded from redundant switch due to the packet drops for VL ID 0x2A01 at main switch with probability 20 %. The number of packets received from main channel is reduced by 20 % while the total number of packets states no packet loss at the end. Finally, the corrupted packets in the camera end system cannot be recovered and cause losses for 20 % of packets.

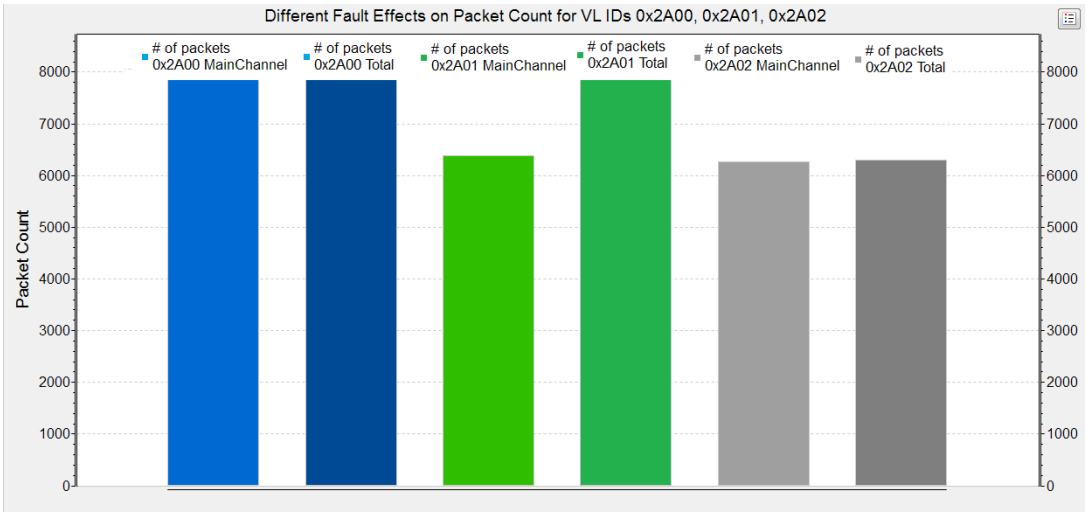

Figure 4.15: Effects of Fault on Packet Count for VL IDs 0x2A00, 0x2A01, 0x2A02 with No Packet Loss, Packet Loss at Switch, Packet Loss at End System Respectively

4.3.2.3 Traffic Burst Effects on AFDX

The previous simulations for AFDX are performed under assumption that the startup time of nodes are randomly distributed in order to prevent switches from traffic burst. Moreover, the different packets generated from the same source is also not synchronized to improve the status of TX queues in the end systems. Yet, it is important to push the limits of AFDX to find the worst case response time. Therefore, additional simulations with different configurations are performed and the results are demonstrated in Table 4.14.

First, it is assumed that all end systems are synchronized and started to operate at t_0. In other words, all periodic packets are generated at the same time and transmitted from end systems. Different configurations of periodic packets do not affect the results of sporadic packets since they are randomly distributed. Due to the burst in switches, end to end latency results of periodic traffic are affected poorly. Nevertheless, it is notable that the maximum latency of periodic messages is not shifted significantly since the origin of this maximum latency value of 3.7 ms is from the message with VL ID 0x1900 regulated by BAG mechanism of AFDX as explained previously. The configuration of synchronization in-node is the case where end sys-

Startup Configuration	Average Latency	Maximum Latency
All Synchronized at t_0	Periodic 0.480 ms Sporadic 1.260 ms	Periodic 3.816 ms Sporadic 23.491 ms
Synchronization In-Node	Periodic: 0.395 ms Sporadic 1.211 ms	Periodic 3.824 ms Sporadic 22.522 ms
No Synchronization (Default Configuration)	Periodic 0.335 ms Sporadic 1.210 ms	Periodic 3.720 ms Sporadic 24.118 ms

Table 4.14: AFDX End-To-End Latency Results under Different Start Time Configuration of End Systems

tems have different startup time, but each of them is able to generate their internal periodic packets synchronously. It performs slightly better but still worse than the default configuration.

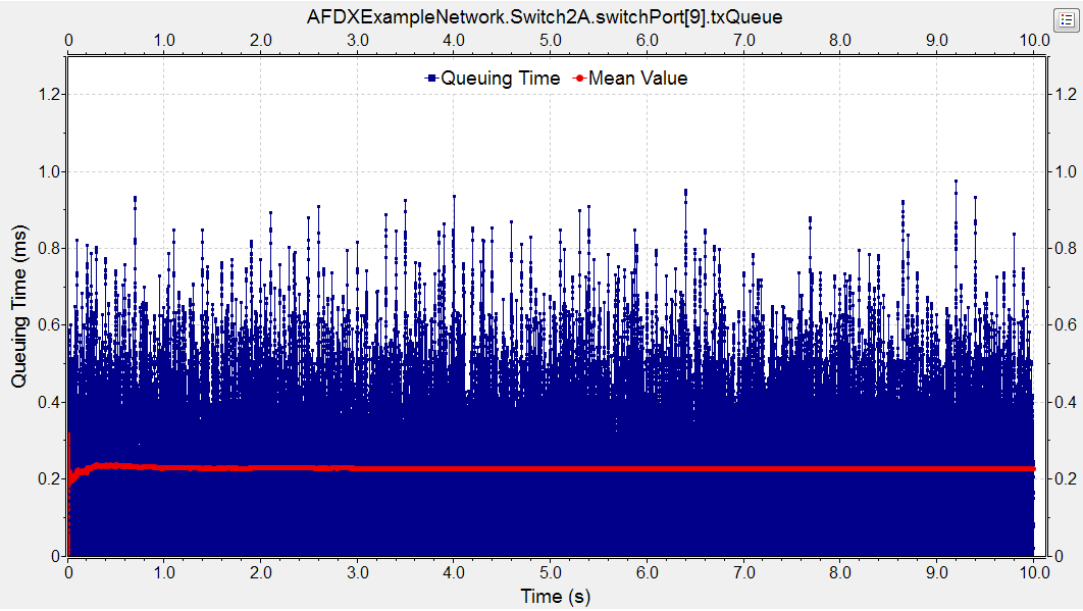

Figure 4.16: Plotted Time-varying TX Queuing Delay of AFDX Switch-2 Port-9 at Traffic Burst

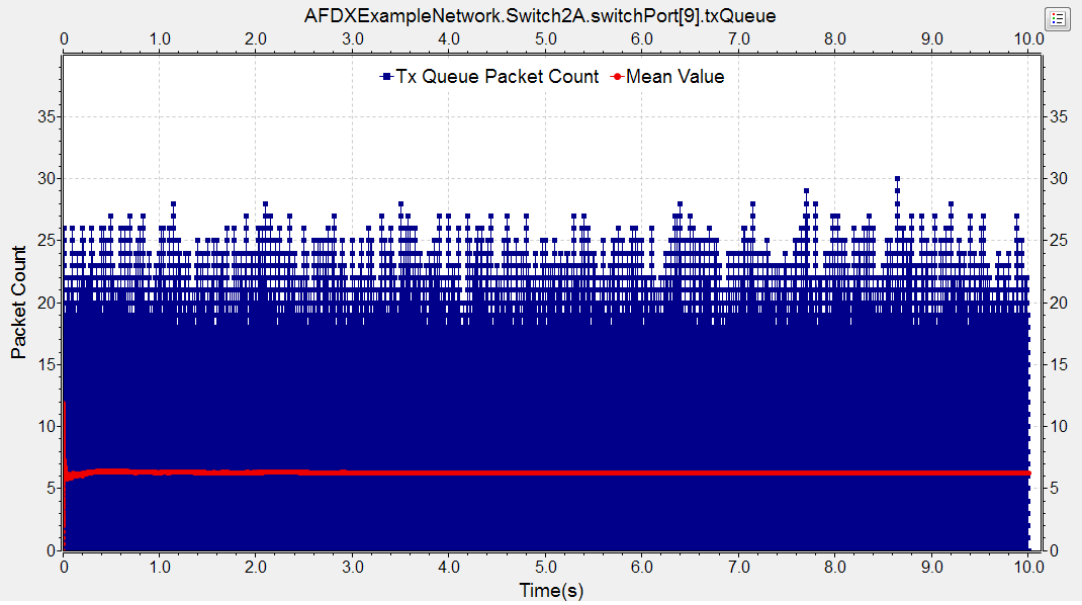

Figure 4.17: Plotted Time-varying TX Queue Length of AFDX Switch-2 Port-9 at Traffic Burst

One remarkable affect of burst could be observed in TX queue of Switch-2 Port-9. It is the most busy port for the given message set in Table 4.6 and the routing configurations in Table 4.4 and Table 4.5. To this end, length of the queue and queuing delay induced to the packets are obtained after the simulation with all synchronized configuration, i.e. periodic packets are transmitted at t_0. It is shown in Fig. 4.16 and 4.17 that the queuing delay at Switch-2 Port-9 could be as much as 1 ms whilst the average is above 0.2 ms. The maximum and average queue length goes up to 30 and 6 respectively.

CHAPTER 5

CONCLUSION

The thesis proposes a novel protocol, Shared Queue based Dynamic Slot Reservation (SQDSR), that is a promising alternative to current prevalent real time communication architectures. It is designed on the base of Ethernet broadcast, where the medium is a half duplex shared bus. SQDSR offers time slots for the transmission of both periodic and sporadic messages in repeated cycles. Whereas the time slots for periodic messages are assigned at design time, SQDSR keeps a global queue that is shared among nodes through broadcast notifications for sporadic messages. The shared queue (SQ) is updated based on special control slots and enables dynamic slot allocation for the idle time intervals at the end of each cycle. Since all nodes share exactly the same queue, collisions on the bus are avoided, which enables us to eliminate non-deterministic nature of CSMA/CD on standard Ethernet.

The proposed protocol includes fault-tolerance features, which are developed based on a detailed analysis of existing fault tolerant real time architectures. Considering the advantages and drawbacks of these architectures including both fieldbus based and Ethernet based protocols, it was seen that there is no architecture that satisfies hard real time, reliability and compatibility without any specialized hardware, suffering from single point of failure (SPoF) and requiring a commercial license. In order to resolve the shortcomings of the current architectures, SQDSR is introduced with detailed specifications including state machine, timing analysis, frame format, fault handling mechanisms and time synchronization. Specifically, SQDSR realizes an efficient implementation of the IEEE 1588 precision time protocol (PTP) to synchronize the clocks of nodes with hardware time stamping with an expected accuracy below 1 μs.

In order to evaluate the performance of SQDSR, the protocol was modelled and simulated using the INET framework of the OMNeT++ network simulator, which provides Ethernet protocols and modules. SQDSR was developed and located at the link layer of the `EtherHost` module of INET. The simulations were performed with a messages from a real application, having both periodic and sporadic traffic types. For the purpose of comparison, AFDX (Avionics Full-Duplex Switched Ethernet) was selected as a benchmarking protocol since it is widely employed in real time networks and provides fast and reliable communication with full duplex switched Ethernet. An existing model of AFDX in OMNeT++ was updated in order to fully cover all the relevant specifications of AFDX for a realistic simulation.

The same message traffic was applied to both SQDSR and AFDX models. Several simulations were conducted with different configurations. The simulation results justify that SQDSR performs better for periodic messages in terms of end-to-end latency. The additional delay component from queuing and processing time of switches worsens the performance of AFDX. It was also seen that the performances for sporadic messages is quite close if the message set is modified such that the nodes sending sporadic messages also have periodic messages. The add-on periodic messages provide static slots to these nodes, which are the notification points of sporadic packets. Otherwise the only way to declare them is the control slots repeated at the beginning of high level cycles.

Furthermore, the simulations test the fault tolerance performance of both SQDSR and AFDX models. SQDSR is expected to handle any faults with the retransmission mechanism in each cycle. A frame containing the information of erroneous packets is sent in the scope of the retransmission mechanism to notify the nodes about frames that need to be retransmitted. The nodes who are responsible for the retransmission of packets, react accordingly based on the order in the notification frame. It was observed that SQDSR handles all the erroneous packets in the retransmission section such that there is no packet lost at the end. On the other hand, AFDX provides dual redundant switches to cope with any failure. While this approach enables to save packets that are lost or corrupted in one of the switches, it does not resolve any faults that could be occurred in end systems. Since AFDX has no retransmission procedure, it is likely to lose some of the packets due to the failures.

The thesis works justify that SQDSR fulfills fault tolerant real time communication requirements by means of the properties of reliability, compatibility and low latency. It is free from any SPoF conditions such as switches or master-slave architectures. It is constructed on Ethernet having commercial off the shelf components and no compatibility issues. Furthermore, it showed successful results during simulations with real case data. Next, it could be required to test SQDSR protocol with hardware as a future work. Moreover, it would be better to compare SQDSR with other competitor protocols.

REFERENCES

[1] I. Álvarez, A. Ballesteros, M. Barranco, D. Gessner, S. Djerasevic, and J. Proenza, "Fault Tolerance in Highly Reliable Ethernet-Based Industrial Systems," *Proceedings of the IEEE*, vol. PP, pp. 1–34, 05 2019.

[2] D. Gwaltney and J. Briscoe, "Comparison of communication architectures for spacecraft modular avionics systems," 01 2006.

[3] P. Pedreiras, P. Gai, L. Almeida, and G. Buttazzo, "FTT-Ethernet: A Flexible Real-Time Communication Protocol That Supports Dynamic QoS Management on Ethernet-Based Systems," *Industrial Informatics, IEEE Transactions on*, vol. 1, pp. 162 – 172, 09 2005.

[4] O. Ltd., "Omnet++ (5.6.1)," 2020-02-10.

[5] O. Ltd., "Inet framework (4.1.2)," 2020-02-10.

[6] ARINC, *Aircraft Data Network Part 7 Avionics Full-Duplex Switched Ethernet Network*, 09 2009.

[7] P. Pendyala and V. S. R. Pasupureddi, "100-Mb/s enhanced data rate MIL-STD-1553B controller in 65-nm CMOS technology," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 6, pp. 2917–2929, 2016.

[8] "Review and Rationale of MIL-STD-1553 A and B," tech. rep., Data Device Corporation.

[9] E. A. Sari Germanos, Wolfgang Seiss, "Synchronizing Mechatronic Systems in Real Time Using FPGAs and Industrial Ethernet Communications," 2015. ALTERA White Paper WP-01249-1.0.

[10] P. Danielis, J. Skodzik, V. Altmann, E. B. Schweissguth, F. Golatowski, D. Timmermann, and J. Schacht, "Survey on real-time communication via ethernet in industrial automation environments," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2014.

[11] "Ethernet Powerlink Basics," tech. rep., Ethernet POWERLINK Standardization Group, 2018.

[12] "Industrial Ethernet Facts," tech. rep., Ethernet POWERLINK Standardization Group, 2016.

[13] "EtherCAT – The Ethernet Fieldbus Brochure," tech. rep., EtherCAT Technology Group, 2020.

[14] "Time-Sensitive Networking: A Technical Introduction," tech. rep., Cisco, 2017.

[15] S. Brooks and E. Uludag, "Time-Sensitive Networking: From Theory to Implementation in Industrial Automation," tech. rep., Intel, TTTech.

[16] H.-T. Lim, D. Herrscher, M. Waltl, and F. Chaari, "Performance Analysis of the IEEE 802.1 Ethernet Audio/Video Bridging Standard," pp. 27–36, 03 2012.

[17] N. Rejeb, A. K. Ben Salem, and S. Ben Saoud, "AFDX simulation based on TTEthernet model under OMNeT++," in *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, pp. 423–429, 2017.

[18] R. Alena, J. Ossenfort, K. Laws, A. Goforth, and F. Figueroa, "Communications for Integrated Modular Avionics," pp. 1 – 18, 04 2007.

[19] T. Steinbach, H.-T. Lim, F. Korf, T. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)," 09 2012.

[20] M. Flajslik and M. Rosenblum, "Network interface design for low latency request-response protocols," pp. 333–346, 06 2013.

[21] Z. Idrees, J. Granados, Y. Sun, S. Latif, l. Gong, Z. Zou, and L. Zheng, "IEEE 1588 for Clock Synchronization in Industrial IoT and Related Applications: A Review on Contributing Technologies, Protocols and Enhancement Methodologies," *IEEE Access*, vol. PP, pp. 1–1, 08 2020.

[22] B. Vattikonda, G. Porter, A. Vahdat, and A. Snoeren, "Practical TDMA for datacenter Ethernet," *EuroSys'12 - Proceedings of the EuroSys 2012 Conference*, 04 2012.

[23] R. Hornig and A. Varga, “Avionics Full-Duplex Switched Ethernet for OMNeT++.” https://github.com/omnetpp-models/afdx, 2017.

[24] D. M. Lane, “Confidence Interval on the Mean.”

[25] Cisco, “Fundamentals of Digital Video,” Sep 2014.