CRANFIELD UNIVERSITY

TAO FENG

# ETHERNET-BASED AFDX SIMULATION AND TIME DELAY ANALYSIS

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING

MSc by Research Thesis
Academic Year: 2015 - 2016

Supervisor: Dr Huamin Jia
February 2016

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING

MSc by Research Thesis

Academic Year 2015 - 2016

TAO FENG

# ETHERNET-BASED AFDX SIMULATION AND TIME DELAY ANALYSIS

Supervisor: Dr Huamin Jia
February 2016

# ABSTRACT

Nowadays, new civilian aircraft have applied new technology and the amount of embedded systems and functions raised. Traditional avionics data buses design can't meet the new transmission requirements regarding weight and complexity due to the number of needed buses. On the other hand, Avionics Full Duplex Switched Ethernet (AFDX) with sufficient bandwidth and guaranteed services is considered as the next generation of avionics data bus. One of the important issues in Avionics Full Duplex Switched Ethernet is to ensure the data total time delay to meet the requirements of the safety-critical systems on aircraft such as flight control system.

This research aims at developing an AFDX time delay model which can be used to analyse the total time delay of the AFDX network. By applying network calculus approach, both (σ,ρ) model and Generic Cell Rate Algorithm (GCRA) model are introduced. For tighter time-delay result, GCRA model is applied. Meanwhile, the current AFDX network simulation platform, FACADE, will be enhanced by adding new functions. Moreover, avionics application simulation modules are developed to exchange data with FACADE. The total time delay analysis will be performed on the improved FACADE to validate this AFDX network simulation platform in several scenarios. Moreover, each scenario is appropriated to study the association between total time delay performance and individual variable.

The results from updated FACADE reflect the correlation between total time delay and certain variables. Larger BAG and more switches between source and destination end systems introduce larger total time delay while $L_{max}$ could also affect the total time delay. However, the results illustrate that the total time delays from updated FACADE are much larger than GCRA time delay model which could up to 10 times which indicates that this updated FACADE needs further improvement.



Keywords:

Avionics Full-Duplex Switched Ethernet, Network Calculus, Arrival Curve, Service Curve, GCRA, Total Time Delay

# ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisor, Dr. Huamin Jia, for his constant support and guidance to my research. Without his advice, my research could not be completed successfully.

Some people help me with my research. I would like to express my appreciation to Mr. Baochua Wu, Mr. Qingming Song, Mr. Yu Lu, Mr. Chaoqun Chen, Mr. Jian Zhang, Mr. Dexin Xu, Mr. Yang Guo, Ms. Xiaojie Zeng, Ms. Lijuan Sun, Ms. Wenjing Wang, for their generous help either on study and living during the research.

I would like to thank COMAC (Commercial Aircraft Corporation of China) for giving me this opportunity to undertake this research, as well as the constant support and help. Special thanks also to Cranfield University, for giving me the chance to promote myself.

Finally, I deeply appreciate my wife, my family and friends for having supported me at all the times.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

LIST OF ABBREVIATIONS

| | |
|---|---|
| ADRA | Avionics Data Reception Application |
| ADTA | Avionics Data Transmission Application |
| AFDX | Avionics Full Duplex Switched Ethernet |
| API | Application Programming Interface |
| ARINC | Aeronautical Radio, Incorporated |
| BAG | Band Allocated Gap |
| BIP | Behaviour, Interaction, Priority |
| CRC | Cyclic Redundancy Code |
| CSMA/CD | Carrier Sense Multiple Access with Collision Detection |
| DSPN | Deterministic and Stochastic Petri Nets |
| FIFO | First In First Out |
| GCRA | Generic Cell Rate Algorithm |
| GPS | Generalized Processor Sharing |
| IEEE | Institute of Electrical and Electronics Engineers |
| IMA | Integrated Modular Avionics |
| IP | Internet Protocol |
| LAN | Local Area Network |
| LRU | Line-replaceable Unit |
| MAC | Media Access Control |
| RTA | Response Time Analysis |
| UDP | User Datagram Protocol |

# 1 INTRODUCTION

## 1.1 Background and Motivation

Since the demands of transportation growing rapidly, new civilian aircraft are designed and manufactured. These new aircraft have applied new technology and the amount of embedded systems and functions are raise. IMA (Integrated Modular Avionics) at platform level and communication multiplexing at network level have been proposed for keeping both system capacity and maintainability. Traditional avionics data buses such as ARINC 429 and MIL-STD-1553 with limited bandwidth can't meet the data exchange capacity of those new civilian aircraft.

Avionics Full Switched Ethernet (AFDX), also known as ARINC 664 P7, has been widely applied in modern aircraft, such as A380 and B787. With sufficient bandwidth (100 times larger than MIL-STD-1553), low cost and redundancy management, AFDX is considered as the next generation avionics data bus.

AFDX network multiplexes huge amounts of data flows over a full duplex switched Ethernet. All these data flows have to compete for sharing network resource which makes the data exchange behaviour uncertain. AFDX introduces specific mechanisms such as the virtual link to guarantee the deterministic of data communication behaviour. Upper bound time delay could be employed to prove this determinism. Moreover, time delay reflects the transmission capacity of the network. Thus, studies on AFDX time delay should be conducted.

It is not common for every researcher studies the AFDX network performance on industrial AFDX network. Simulation approaches could be proposed since it simulates the AFDX network environment with conventional software and hardware. Thus, the cost of the simulation is acceptable. With the simulation environment, studies of the AFDX network could be quickly conducted. Moreover, it could be utilised by the AFDX system designers for AFDX network design. Moreover, the simulation environment could help AFDX system

designers during their design period. The duration of design could be reduced and the cost could be decreased.

Recently, studies on AFDX time delay analyses have several approaches. Theoretical approaches propose mathematical techniques to obtain the tighter AFDX network time delay. Meantime, the simulation approaches are used for verification and give more accurate results. Simulation with real-time software which excluded network hardware is proposed to simulate certain scenario. Without hardware influence considering, this simulation is not similar to the industrial environment. Programming software simulation with network hardware, on the other hand, tries to simulate the industrial environment which is more realistic. Till now, few researchers focus on AFDX simulation with software implementation combined network hardware.

In this project, the time delay model of AFDX network is introduced by applying Network Calculus. Then programming software simulation with network hardware involved AFDX network simulation platform FACADE is developed. Moreover, this simulation platform is suitable for AFDX network experiment purposes. Besides, the total time delay of the AFDX network and the associations between total time delay and several variables are deduced and the comparison between calculus and experiment delay is made. In conclusion, this project plans to benefit the design of avionics data bus by offering designers possible approaches to optimise the performance of total time delay, in the academic or industrial area.

## 1.2 Research Objectives

This research aims to develop methods and platform to analyse and evaluate AFDX system design requirements and network time delays to provide AFDX system designer with a design verification tool. Several measurable research objectives have been identified and outlined as follows:

a) Modelling the AFDX time delay.
b) Improving the current AFDX network simulation platform, FACADE by introducing three new functions, including the loss data detect function,

invalid data detect function and communication time record function. Developing avionics application simulation modules which could communicate with FACADE platform.

c) Testing the improved FACADE simulation platform.
d) Detecting the associations between the total time delay and certain variables.

## 1.3 Thesis Structure

This project consists of five main phases which are background knowledge comprehension, AFDX time delay model by Network Calculus, platform understanding and modifications, experiments design and execution as well as data collection and analysis. These steps act as the time sequence. All these phases are listed below:

**Phase 1: Background knowledge comprehension**

In this phase, contextual comprehension of the research and related knowledge should be studied first. AFDX network standard and its related standard should be investigated elaborative. It is essential for the researcher to understand AFDX network characteristics as well as data exchange behaviour. Since the simulation bases on the AFDX standard, a literature review on AFDX network is conducted. After that, the research approaches proposed by other researchers were reviewed.

**Phase 2: AFDX time delay model by Network Calculus**

This section aims to study the time delay model of the AFDX network using Network Calculus. The whole AFDX network consists of three main components which are end system, AFDX switch and physical propagation. In this project, traffic regulator in end system is considered as powerful enough which will not impose extra delay to transmission. The primary analysis part of the switch is scheduler. Moreover, the physical propagation can be assumed upper bound by some constant value. All these three delays compose the total delay time of the AFDX network.

**Phase 3: Platform understanding and modifications**

Since the simulation platform framework architecture proposed in this research is as same as the previous work, comprehension of this simulation platform should be carried out instantly. At first, the theses written by previous researchers are studied. The requirement analyses contained in those articles help the author to understand the initial motivation of this platform easily. The detailed framework description makes the author comprehend the grand architecture effortlessly. Certain modifications have been made to develop this platform suitable for the following experiment after containing the previous simulation platform. The Af module has been modified into an AFDX data packets reception interface of the AFDX network while Raf module has been changed into an AFDX data packets transmission interface of the AFDX network. The particular part of the data has been proposed for data check function. Time record function and data packet counting function are also invited for experiment purpose. Moreover, two new modules are implemented as avionics application on the end system. One is used to send AFDX data while the other module is proposed to receive AFDX data.

**Phase 4: Experiments design and execution**

In this phase, the experiments which focus on computing total time delay and associations between target variables and total time delay are design first. Several variables which may influence the total time delay performance of AFDX network are listed as below: $l_{max}$, BAG of virtual link, the amount of destination end systems in one specific virtual link, the amount of virtual links in one AFDX network and the amount of switches between source end system and target destination end system. After design, each experiment will be executed and the experiment issues will be stated as well as the experiment data.

**Phase 5: Data collection and analysis.**

This phase is employed to collect and analyse the data of those five experiments. Firstly, data are gathered from each experiment separately. Then those data will be examined respectively. In this phase, the associations between total time delay and $l_{max}$, BAG, the amount of destination end systems

in one particular virtual link, the amount of virtual links in one AFDX network and the amount of switches between source end system and target destination end system will be studied. The normal test is executed to detect whether student's t-test can analyse the obtained data. The scatter chart, as well as box chart, are applied to the data analysis section. The comparison between calculus total time delay and experiment total time delay are carried out.

As represent in Figure 1-1, there are six chapters in this thesis including chapter 1. In chapter 2, studies on AFDX network and recently related studies are summarised. Chapter 3 invites the Network Calculus and then time delay model of the AFDX network will be analysed. The detailed design of AFDX network simulation platform and avionics application simulation modules are described in chapter 4. Chapter 5 focuses on the design of the experiments which are utilised to detect the associations between total time delay and several variables. Moreover, chapter 5 represents the experiments states, analysis and deduction of those obtained data. The comparison between calculus delay and experiment delay is carried out. In chapter 6, the key findings and future work are illustrated and concluded.

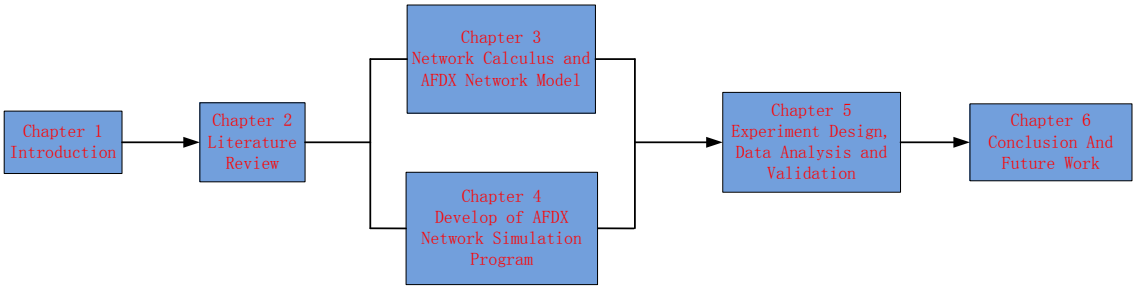


**Figure 1-1 Thesis Structure**

## 1.4 Summary

In this chapter, the background and motivation of this project have been introduced first. AFDX is considered as the next generation of avionics data bus, the time delay which reflects the performance of AFDX network should be studied. Simulation of AFDX network makes research more easily to be

conducted. Secondly, the objectives of this project are represented. Finally, the structure of this thesis is listed.

# 2 LITERATURE REVIEW

## 2.1 AFDX Introduction

AFDX has already applied on Airbus A380, A350, Boeing 787, COMAC ARJ21, and other aircraft. ARINC 664 standards, also known as AFDX (short for Avionics Full Duplex Switched Ethernet), has been published in 2005. AFDX [1][8][9] transmission rate up to 100Mb/s which is 1000 times faster than its predecessor ARINC 429. The deterministic behaviour and redundant architecture meet the avionics requirement. Nowadays, most future civil transport aircraft are going to adopt this standard communication network [2].

IEEE 802.3 is a LAN (local area network) technology which provides network access service. Devices connect to switches or other infrastructure devices by various types of cables. AFDX standard is based on Ethernet technologies especially IEEE 802.3. As the commercial 802.3 hardware are low cost, AFDX tries to use them as much as possible. Although IEEE 802.3 provides high exchange communication capacity, it could not meet the safety-critical avionics systems' requirements due to its uncertain data exchange behaviour and frame loss. AFDX has introduced a new mechanism to overcome these problems which IEEE 802.3 has. To determine the communication behaviour, AFDX adds the virtual link. The virtual link with a maximum dedicated bandwidth provides guaranteed maximum network transmission delay. Redundancy management is proposed to solve the frame loss issue. For each virtual link, frames are transmitted across both redundant networks. Meanwhile, the "First Valid wins" policy is applied. The first frame reaches either two networks is accepted while the second frame is just discarded. This redundancy management guarantees the data exchange when any one network component fails.

AFDX network consists of three elements which are end system, AFDX switch and virtual link.

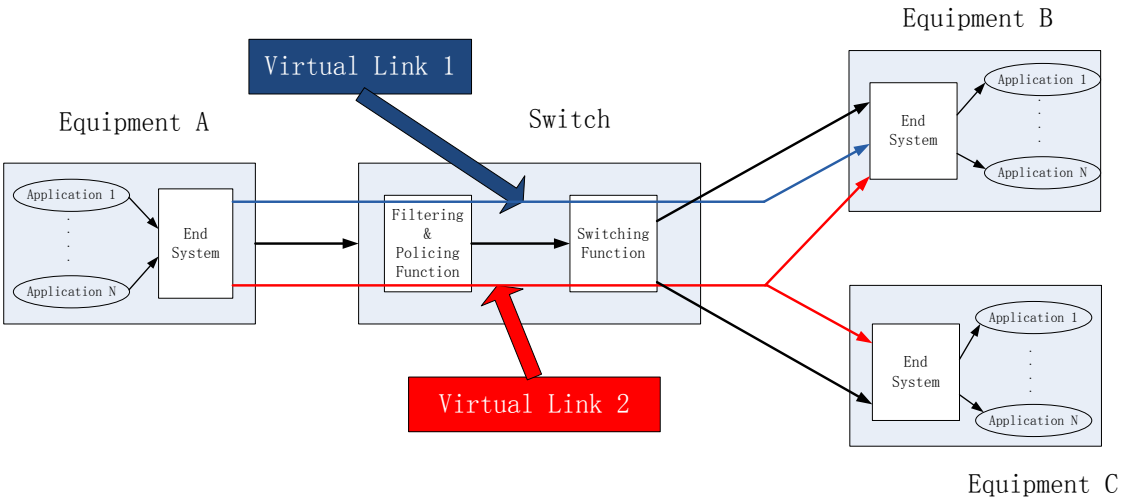


**Figure 2-1 Illustration of AFDX Network**

Figure 2-1 represents these three elements. End system is working as a source or receptor of AFDX data on the network. Source end system receives data from source avionics application and transfers those data to destination end system. After destination avionics application receives data from destination end system, the transmission finished. During the transmission, data are transmitting on virtual links between source and destination end systems. AFDX switch is proposed to connect end systems to network, receive, handle and transmit the data to destination end system.

### 2.1.1 End System

End system (or avionic computing LRUs) exchanges data (messages) based on the concept of virtual links with traffic shaping, redundancy management, integrity checking and routing provided by AFDX switches using configuration tables. Avionics application sends the data to end system with unregulated flows of post fragmented packets. Traffic regulator in end system shapes those uncontrolled flows into regulated flows according to Bandwidth Allocation GAP of each virtual link.

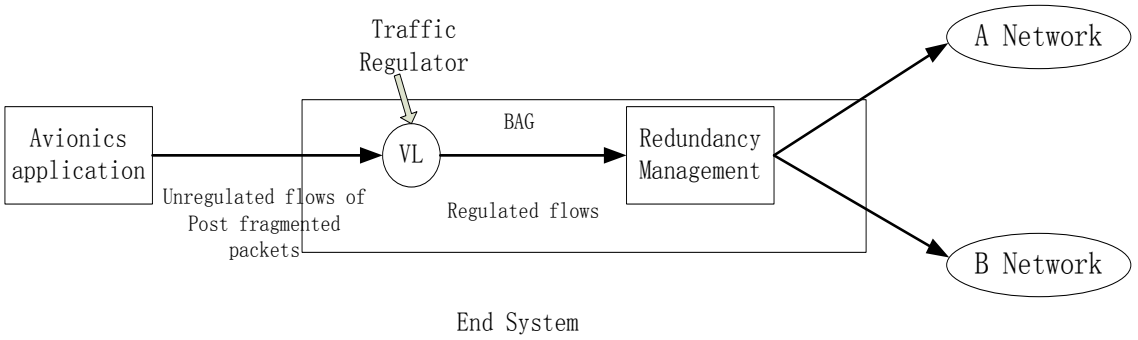


**Figure 2-2 Source End System**

Bandwidth Allocation GAP, also known as BAG, is the minimum time interval in the millisecond of the first bit of two consecutive frames of the same virtual link. BAG is used for managing data flow. (See Figure 2-3). BAG values equal to $2^k$ ms interval, (k is an integer from 0 to 7).

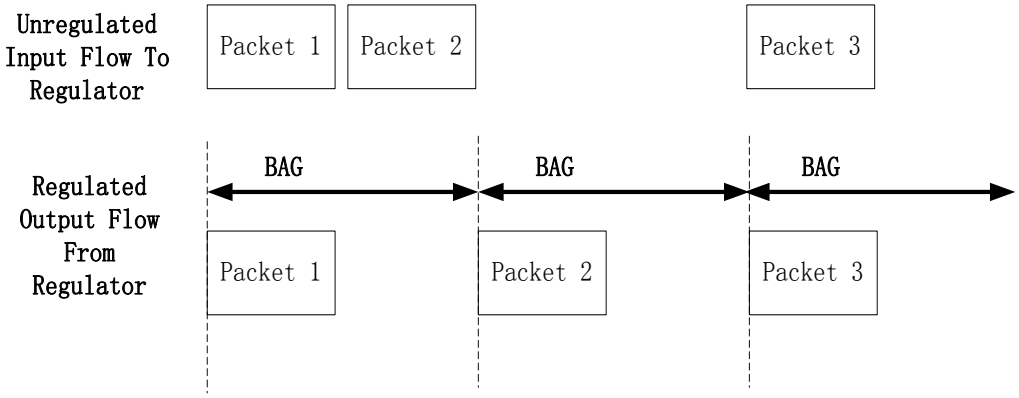


**Figure 2-3 Unregulated and Regulated Flow**

After shaping, each flow will send no more than one frame in every interval of BAG. Then each frame in the flow will be forwarded to both networks by redundancy management. By using redundancy management, data communication could continue even one complete network lost.

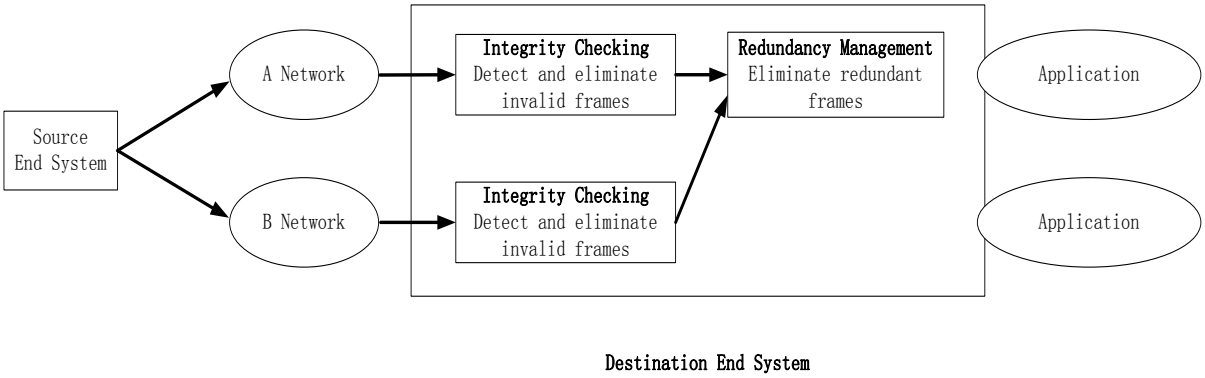


**Figure 2-4 Destination End System**

After data frames across network and reach destination end system, integrity checking will first detect and eliminate invalid frames. Then the “First valid wins” policy is introduced which means that the first valid data frame reaches the redundancy management will be accepted and transferred to avionics application.

### 2.1.2 Virtual Links

A virtual link has these characters:

A virtual link is a logical connection which connects one end system to one or more end systems. Network designer can configure the upper bound bandwidth of each virtual link [2].

The bandwidth of a virtual link can be allocated by end system and would not be affected by other partitions which use this virtual link.

Each virtual link should not exceed its upper bound size and transmission rate which will be checked by the switch. These parameters store in the switch, follow by the switch and end systems. The switch can also change virtual link priorities to secure the performance of time-sensitive messages.

Ethernet switch receives incoming frames from the input port and routes them to output port base on the destination address in Ethernet transmission. In AFDX network, virtual link ID is invented as a unique mark of a virtual link (See Figure 2-5). This mark replaces source address, target address(es) and data

content amount of the frame which stores in switch's configuration table. Virtual link id represents the source of the virtual link where the frame comes.

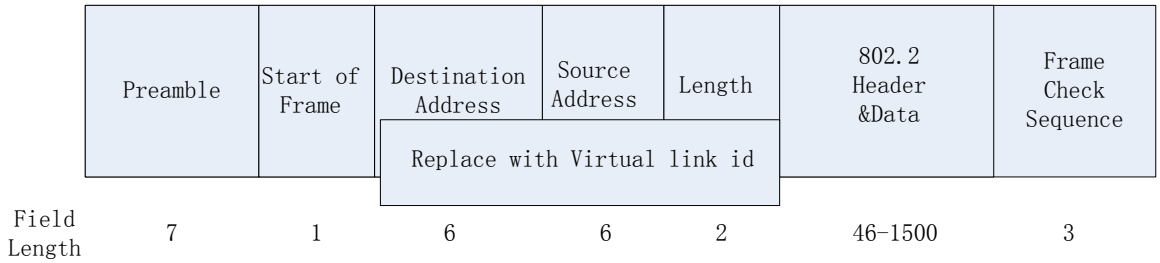


**Figure 2-5 AFDX Network Packet Format**

A virtual link is a logical connection for network data in transmission– see Figure 2-6. The upper bound of latency and jitter are guaranteed due to the network bandwidth is allocated. The virtual link connects source end system and destination end system(s) together by using the switch which means that an end system may be designed only to receive VLs and not transmit virtual links, or on the contrary. In that way, an end system could have zero virtual links, neither originates or receives. The virtual link sends frames for end system. A virtual link can only have one end system as the source, but one or more destination end systems are permitted.

The sub-virtual link is introduced to use the virtual link more efficient. One virtual link could possess up to 4 sub-virtual links. Each sub-virtual link follows First In, First Out regulation. Each First In, First Out queue is accessed by virtual link using round robin algorithm.

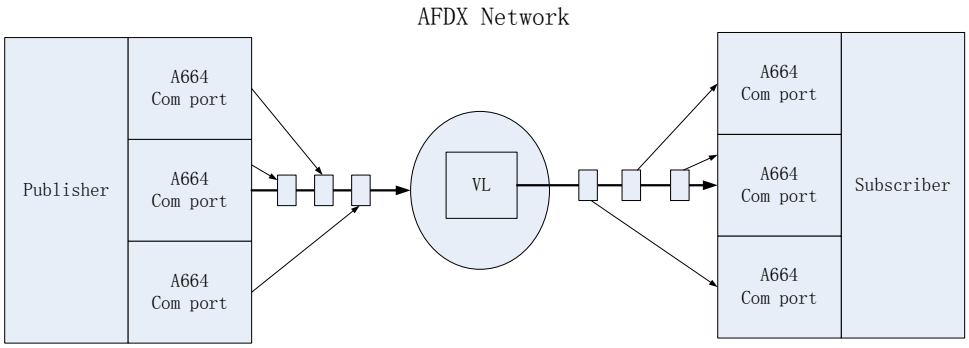


**Figure 2-6 Chart of Virtual Link**

### 2.1.3 AFDX Switch

Based on 802.3 Ethernet techniques, AFDX is a full duplex network. The transmission medium connected each end system is a network cable which contains two pairs of wires. Those two pairs of wires have different functions, one is to send frame (Tx), the other is to receive frame (Rx). (See Figure 2-7)

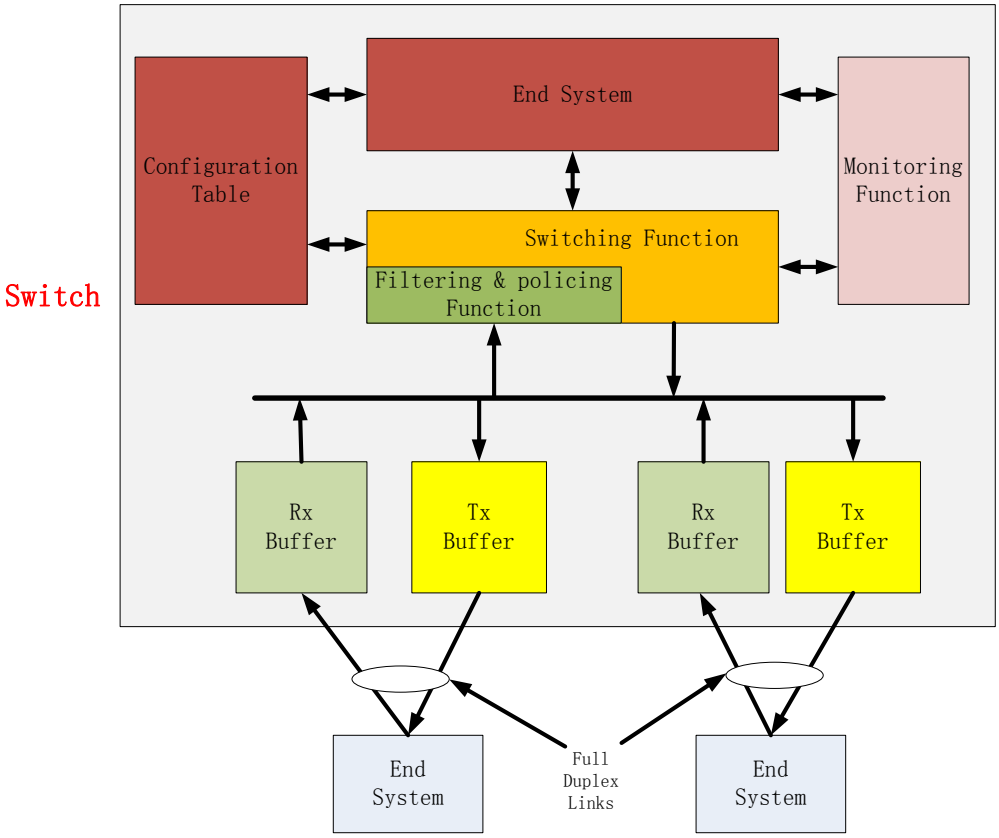


**Figure 2-7 AFDX Switch**

AFDX switch has several blocks. Filtering and policing function filters frames base on rules such as frame integrity, frame length and destination. Switching function routes frames to their destinations through appropriate output ports. The configuration data which controls the switch are stored in the configuration table. End system in AFDX switch is utilised to receive and transfer data. The monitoring function is introduced to monitor and log all operations of the switch.

The switch has two kinds of buffers. Rx buffers are used to store incoming frames while Tx buffers are used to store outgoing frames. Both buffers operate under FIFO mechanism. Switch processor checks all the incoming frames

virtual link IDs sequences. All the sequences should follow the same order as Rx FIFOs. These frames whose sequences are correct will be transferred to the proper Tx buffers, waiting for transmission.

To avoid contention, buffers are used by the switch. Breach of traffic contract might overflow input buffer caused network data loss, is protected by fault containment. Switch buffer allocated to one end system which could manage the contention when two or more end systems send messages to that end system.

Redundancy guarantees a high degree of availability. For instance, cluster switches and two logical connections. CRC, one of the error detection mechanisms, is used to ensure the high integrity. Messages are delayed inevitably due to the process of resolving contention which called as latency.

Switch controls the data traffic in the data bus, following the configuration in the configuration table. As the subsystems have their specific requirements of the network performance, the network designer has to meet these demands which are proposed by avionics application designer during their design. The configuration table considered as an avionics application designer's requirements set for network performance to supply the avionics applications. Network designer expresses all the network performances designed for the avionics applications in the configuration table. As the switch can read the configuration table, it is possible for network performances to meet the demands of avionics applications. This mechanism achieves the certification for network and avionics applications since the designers monitor data bus performance and collect evidence for identification.

### 2.1.4 Latency and Jitter

Two types of latency, latency in transmission and latency in reception, are defined in ARINC 664 P7.

The definition of latency in transmission is the duration of the last bit of an avionics application data which is ready to be sent by end system and the last bit of the corresponding Ethernet frame transfer to the physical media.

The definition of latency in reception is the duration of the last bit of an avionics application data accepted by the physical media and the last bit of the corresponding data is available to the end system avionics application.

According to ARINC 664 P7, the upper bound latency of the end system in transmission should be less than 150μs+frame delay. The upper bound latency of the end system in reception should be less than 150μs.

The time one end system receives a message from the other end systems could be different, even from the same end system. The upper bound time subtracting lower bound time is Jitter. Several aspects could affect Jitter such as Network topology, hardware, the number of virtual links on a single port. Each virtual link has a maximum allowable jitter which stores in the configuration file.

### 2.1.5 Network Topology

At first, Ethernet was not used on aircraft. It is used for the commercial general purpose, to connect devices, terminals as well as systems together and to communicate with each other, air traffic control systems, weather broadcasting systems and navigation systems for instance. As passengers' entertainment systems aboard on the plane, Ethernet was used to connect each device to one domain for management and data sharing.

In Figure 2-8, the devices are divided into two different domains. These two domains connect into a star topology network by two switches.

From this figure, ARINC 664 P7 terminals comprise the avionics systems which are part of each domain. In each domain, the network is connected to the switch which is called domain switch. End systems in same domain communicate with each other by domain switch. When end systems exchanging data in different domains, data are routed through each domain switch, from source to destination.

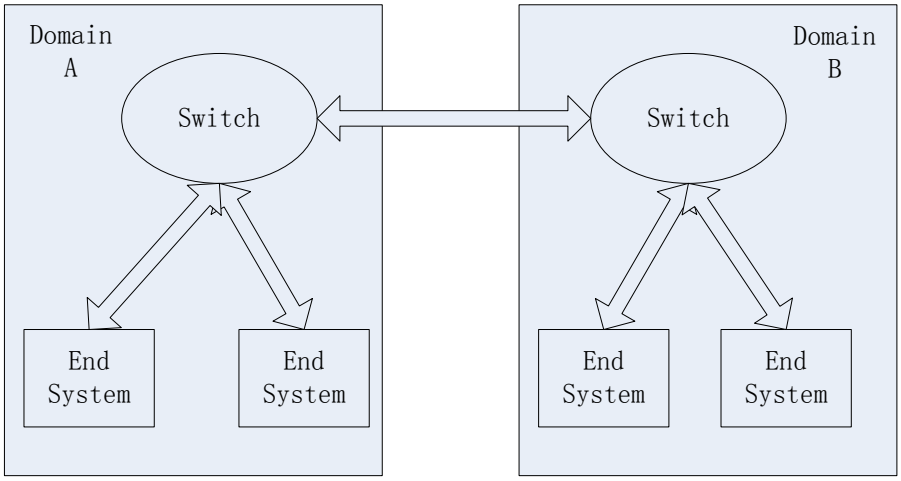


**Figure 2-8 AFDX Star Topology**

## 2.2 AFDX Delay Research Techniques

Several technical methods have been proposed to analyse and evaluate AFDX network delay performances.

Network calculus (or worst case Network Calculus) theory is one of the most common theoretical approaches which is used to obtain end-to-end upper bound delay [4] [5] [6]. The network calculus could provide the mathematical upper bound latency of any network element which possesses queue capability. Moreover, this approach could obtain queue-size bounds as well. As the worst case scenario considers each node crossed by a given flow and calculus maximum possible jitter caused by the previously node, the results obtained by Network calculus is obviously pessimistic [11]. This theory is mainly used for avionics network certification. In the communication network, the arrival curve and service curve are introduced to describe the characteristics of data flows and the scheduler multiplex respectively. A scheduler or multiplexer is a device which receives several input data flows and then forwards those streams into a single line. Thus, the arrival curve is used to represent virtual links in [7]. The network calculus approach obtains a pessimistic upper bound delay which can hardly be reached. To get a tighter computation, several optimizations are taken by researchers. The ‘Group’ concept is introduced to Network Calculus to obtain more accurate delay values. Those virtual links which come out from the same multiplexer and transfer into another multiplexer together are treated as ‘Group’.

These virtual links share at least two paths which will not be serialized by the following multiplexers after they have been serialized by the first one. By utilising this concept, less delay spent in multiplexers. Tighter bounds are obtained up to 40% better than Network Calculus without ‘group’ [12]. Periodic Virtual Link, a virtual link sends frames strictly periodically, is also introduced to gain a more precisely upper bound delay [13]. Periodic virtual link is schedulable with the offset which can obtain a reduction of 49% upper bound delay than network calculus approach.

Due to the massive configuration work on the real network, exact stochastic analysis of industrial avionics network is impracticable. Pessimistic stochastic analysis [15], considered as an approximation of the exact stochastic analysis, is applied to solve this issue. The pessimistic stochastic analysis provides a guaranteed upper bound end-to-end delay which is greater than the exact stochastic analysis. Based on the pessimistic assumptions, the stochastic network calculus approach is a pessimistic analysis. This approach provides a probabilistic upper bound delay of a given flow which belongs to a specific network. By comparing the pessimistic upper bound delay with the industrial AFDX network experiment upper bound delay, pessimistic upper bound delay is no more than four times than the experiment one [16].

Trajectory approach [10] is applied to get the upper bound on end-to-end upper bound delay in distributed systems [11]. In Trajectory approach, the switch output port is treated as a process node. For a packet m from flow f, the trajectory approach will identify all the impacts of end-to-end delay on every node this packet cross. This approach is suitable for the scenario for a group of sporadic data flows without considering when the packets arrived. Trajectory approach focuses on the trajectory of a packet in worst case scenario rather than any visited node in the holistic approach. This approach only considers the possible scenario. As a timing analysis approach, trajectory approach concentrates on given flow, trace a packet and construct the packet sequences in each passing node. This approach is used for obtaining the upper bound arrival times for every packet. According to [14], Trajectory approach is more

precious than the Network Calculus. The ‘Group’ concept is also applied in Trajectory approach to obtain a better upper bound delay. By utilizing this concept, 10% better tighter average upper bounds are obtained than the Trajectory approach without ‘group’.

The response time analysis (RTA) technique has been used extensively to schedulability experiment and other characteristics for the diverse real-time operating system [17] [18] [19] [20]. RTA bases on the critical instant and busy period. A critical instant allows detecting the worst response time for the given task. Each task of simultaneous activation from fixed priority scheduling of single processor is a critical instant [21]. Response time represents the end-to-end delay which means the worst-case response time is the upper bound of the end-to-end delay. By detecting the response time of each task in an interval starting at a critical instant as soon as the job been released and comparing the result to the related deadline, schedulability test for the periodic task set can be acquired as well as the sporadic task. The multiplexer in AFDX can be represented as a processor while RTA is introduced to the worse-case data transmitting delay analysis in the real-time operating system [4].

In [22] [23], the Petri Net Theory is proposed to simulate the AFDX network for AFDX modelling and performance analysis. A deterministic and stochastic Petri nets (DSPN) model is achieved. In this model, data frame length is 791 bytes. Three sub models contains in DSPN model are periodic message control sub model, event message sub model and message transmission sub model. The periodic message control and event message represent the end system in AFDX network while the message transmission sub model expresses the AFDX switch. Authors compute the network load, the time delay of this model and believe this model can be applied to AFDX simulation. Since the switch queue end-to-end delay is not considered in this simulation, it still not similar to the realistic AFDX network.

Alur and Dill first introduced the timed automata [24]. It is a finite automata with real and positive variables increasing uniformly with time. A transition has three labels which are a condition on clock values, actions and the new value of clock.

The model checking approach bases on timed automata is proposed to evaluate the performance of the AFDX contains investigating each state of the system as well as detecting an exact worst-case end-to-end delay [25].This approach is used to represent the AFDX network action by time. For the end system, each state of the frame is represented by a node and the transition expresses the transmission action. For switch, each node represents a location in the queue of the transmit port. Time of transition will be upgraded after each action. The delay can be obtained by calculus the deviation of the new value and old value of clock. The model checking approach has detected all the possible states of the system. Thus, authors believe that an exact worst-case end-to-end delay has been obtained as well as its related scenario. The authors also uncertain if this approach can be applied to a realistic AFDX network.

## 2.3 AFDX Simulation Techniques

### 2.3.1 Real-time Software AFDX Network Simulation

Experiments have to be taken to obtain realistic results. Those results are used to verify the designs. Since not every researcher can utilise a realistic AFDX network environment to study the behaviour and performance of the AFDX network, simulations have been carried out to fix this issue.

The Network simulation 2 (NS2) is proposed to simulate the AFDX network [26] [27]. Figure 2-9 represents the simulation model of the AFDX network end system and switch system. NS2 simulates the functions of the end system and AFDX switch. In end system, the NS2 application layer protocol is used as an application procedure queue. By utilising the traffic shaping sub-function, the flow-shaper is achieved on the queue. The multiplexer is realized by the virtual link scheduling algorithm while the de-multiplexer possesses the redundancy and integrity functions. The AFDX switch and virtual link are also obtained by NS2, which shows that using NS2 for AFDX simulation is practicable. In AFDX switch, the filtering and policing function are simulated by the filtering and policing function in NS2. The switch scheduling algorithm is used to simulate the multiplexer and de-multiplexer. A queue with limited delay and allocated

bandwidth is employed to simulate the virtual link. After that, this simulation is proposed to research end-to-end delay performance. The results show that the jitter of this simulation is between -0.5 ms and 0.5 ms which meets the requirement of ARINC 664 P7.

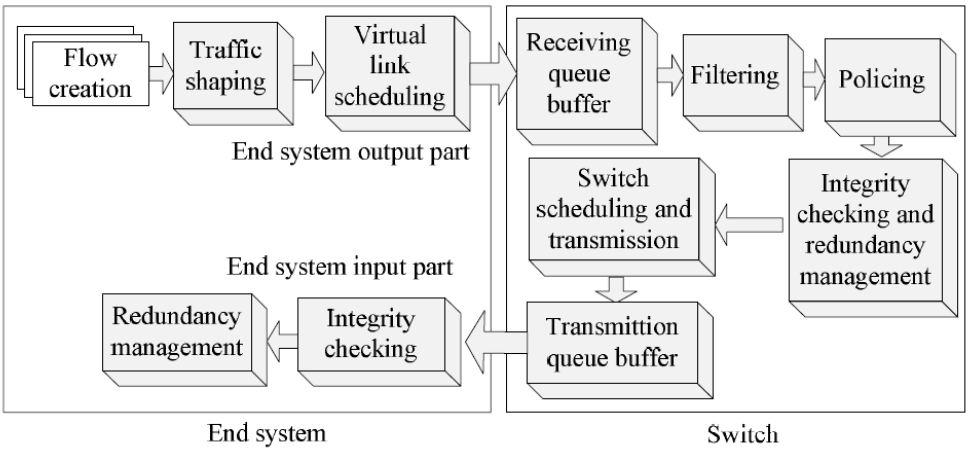


**Figure 2-9 The AFDX End System and Switch System Simulation Model [26]**

The QNAP2 [29] (Queuing Network Analysis Package) is applied to simulate the AFDX network [28]. QNAP2, a commercial package, can be utilised to analyse the performance of queuing network by analytical and simulation approaches. In this software, the end system is represented by four types of queues which are application queues, regulator queues, “Multiplexer” queue and the “DeMultiplexer” queue. The switch is represented by three different queues which are input queues, “CPU” buffer and output queues. The measurements units are also introduced in this simulation. Each frame has placed a time-stamp for obtaining the duration time. This simulation has also been validated which proves that the QNAP2 can be used for AFDX network simulation for end-to-end delay research. The results show that the max delay of simulation is much larger than the mean delay.

The BIP [32] (Behaviour, Interaction, Priority) framework, partially achieved in the IF toolset [30] and the PROMETHEUS tool [31], is proposed for modelling the AFDX network. BIP is a language which can build components from atomic components, connect and priority relations. The atomic components are utilised to build systems. The connects provide possible interactions between systems.

The priority relations select possible interaction. By using BIP, the end system and AFDX switch are simulated. An AFDX experiment architecture is achieved (See Figure 2-10). Several scenarios have also been tested to validate this simulation which proves that this simulation is suitable for the end-to-end delay performance research.

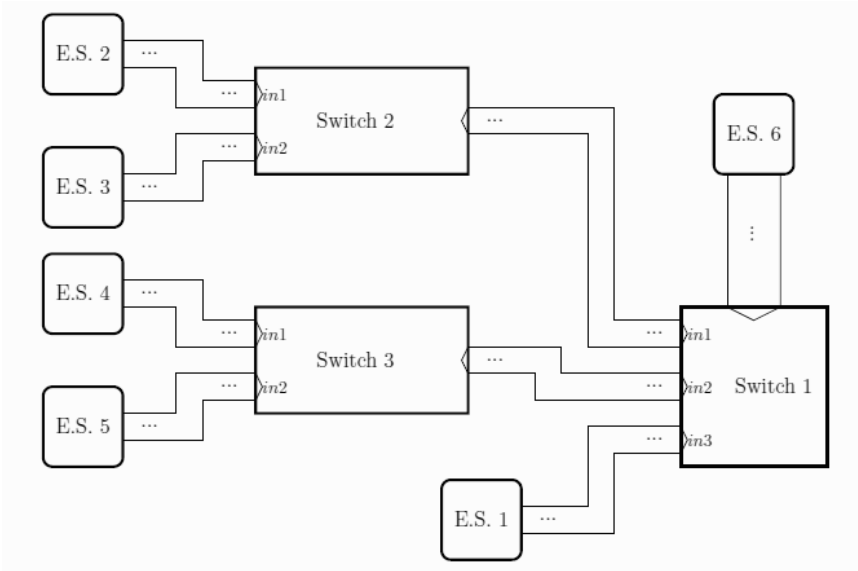


**Figure 2-10 The Architecture of The Experiments [33]**

The MAST suite is suitable for simulating a real-time system [34]. As an event-driven model, MAST can build complex dependence patterns of diverse tasks [36]. This characteristic makes the MAST ideal object-oriented and event-driven architecture for real-time systems. Offset-based techniques are also applied to the MAST for accurate results. Figure 2-11 represents the tools which will be included in the MAST. As the simulation model is a distributed system, several response-time analysis approaches for the processors [47] [48] [49] [50] [51] are proposed and combined with the composition approach [52]. Then the MAST is used to simulate the AFDX network. After obtaining the result by MAST, authors compare their results with the model checking approach, Network Calculus, Network Calculus with grouping, Trajectory approach and optimized Trajectory approach. The comparison represents that the results gain from MAST are only slightly better than Network Calculus and more pessimistic than other approaches [35].

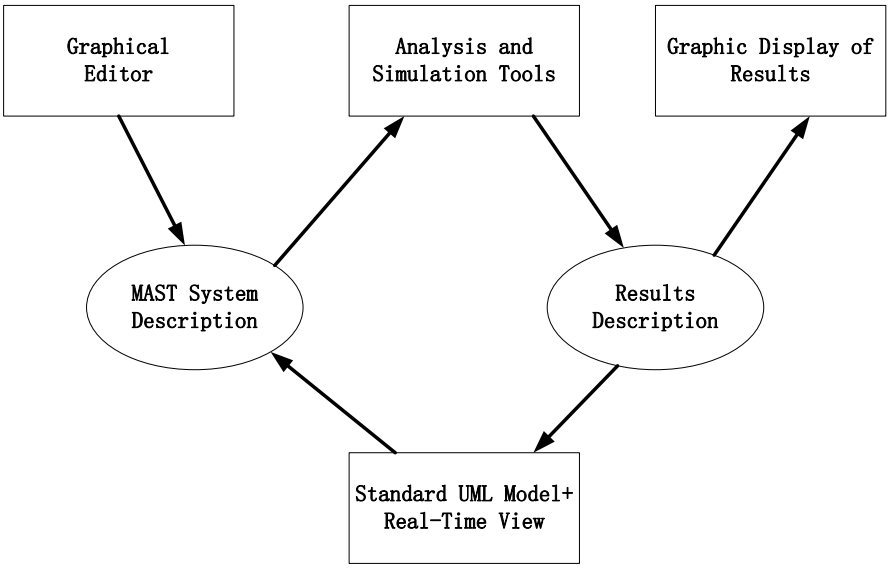


**Figure 2-11 MAST Toolset Environment [34]**

The TrueTime [38] is a real-time network simulation software platform. In [4], authors utilise TrueTime to simulate an AFDX network for evaluating the response time analysis technique. The simulation model is represented in Figure 2-12. Three different scheduling policies proposed to this simulation are BAG-based, rate-based and FCFS schemes. Among these scheduling policies, BAG-based shows the smallest end-to-end delay. In [39], authors utilise the TrueTime to study the Generic Cell Rate Algorithm (GCRA) model which gives a tighter upper bound delay than the (σ,ρ) model.

### 2.3.2 Programming Software AFDX Network Simulation

A programming software simulation of the AFDX end system is established on VxWorks Operating System [40]. This simulation contains the traffic shaper as well as the virtual link scheduler. The source end system is achieved by introduced the transmitting algorithm to Tx process while the destination end system is obtained by introduced the receiving algorithm to Rx process. Then the simulation is carried out on Power PC 7448 and bridge tsi109. According to the experimental results, this software implementation obtains a better jitter performance than the theoretical one which means the simulation is successful. An AFDX emulator is achieved in Ada and verified under the realistic environment by PCs and switches [41]. In this simulation, the scheduler task

and the listener task are introduced for sending data and receiving data respectively. Outbound buffer and inbound buffer are established as virtual links. The outbound buffer has four internal FIFO queues which use as the sub-virtual link. The internal queues can also fragment messages sent by avionics application. Research in [42] uses C++ to simulate an AFDX network for OMNET++. A software named A-Stack [43] is also used as AFDX software simulation platform. This commercial software is certified by DO-178B Level A standard which is a software consideration in the airborne system [44].

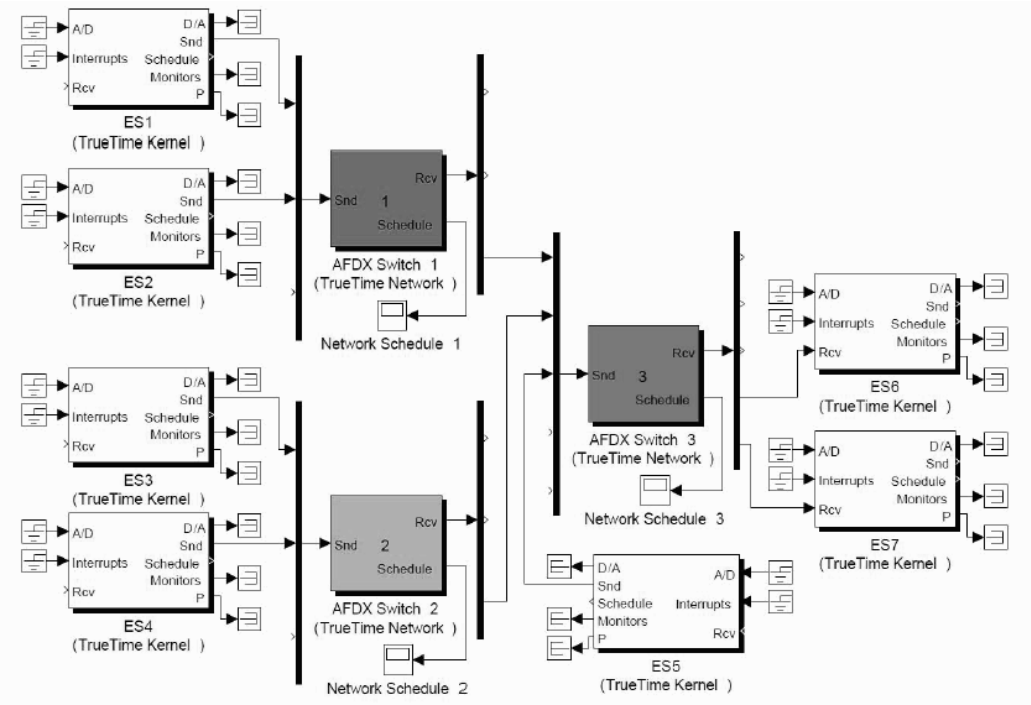


**Figure 2-12 TrueTime Simulation Model [4]**

## 2.4 Summary

This chapter introduces AFDX and related studies. Firstly, AFDX and relative knowledge are listed. Then the research approaches of AFDX have been reviewed. The theoretical approaches are utilised to obtain the mathematical network performances. At the same time, the simulations approaches are used for verification. Simulation with the real-time software which excluded network hardware is not similar to the realistic environment while the programming

software simulation with network hardware is more similar to the real AFDX network.

# 3 Network Calculus Model and AFDX Delay Analysis

## 3.1 Data Flow Features in Data Network Concepts Cumulative Functions

In the Network Calculus, the cumulative function or flow, R(t), is defined as the aggregate of bits which can be seen on the data flows in time interval [0,t]. In the network, a system which could be a switch or an end system handles the input data and sends them with a variable delay. As represented in Figure 3-1, both R(t) and R*(t) are the cumulative functions. R(t), an input function, demonstrates the total input data of the input flow in the time interval [0,t]. R*(t), an output function, represents the entire output data of the output flow in the time interval [0,t].

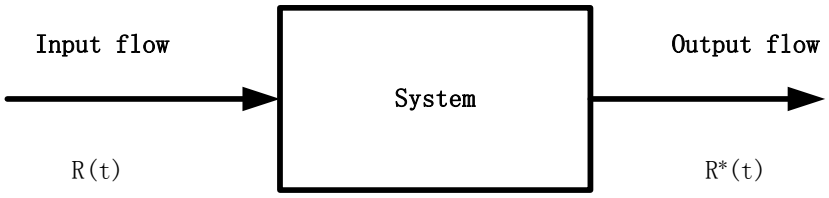


**Figure 3-1 Cumulative Function in Network**

The cumulative function is considered as a wide-sense increasing function. Moreover, when t equals to 0, the cumulative function also equals to 0 as convention. Take Figure 3-1 for instance, R(0)=0 and R*(0)=0. Both discrete and continuous time models can apply this cumulative function. A minimum granularity can always be defined in the realistic world, for instance, bit, word, cell or even packet, hence discrete time with a limited aggregate of values for a cumulative function can be granted. However, a continuous time is more convenient to consider no matter cumulative function is continuous or not. A fluid model is proposed to describe the continuous function while the left or right-continuous function is used to describe an uncontinuous function.

Three types of cumulative functions are listed below. The first function is the discrete time model cumulative function which time t can be represented as $t \in N = \{0, 1, 2, 3,...\}$. The second function is a fluid mode cumulative function

which time t can be represented as t ∈ R+ = [0,+∞) and R is a continuous function. The third function is the general, continuous time model cumulative function which time t can be represented as t ∈ R+ and R is a left- or right-continuous function.

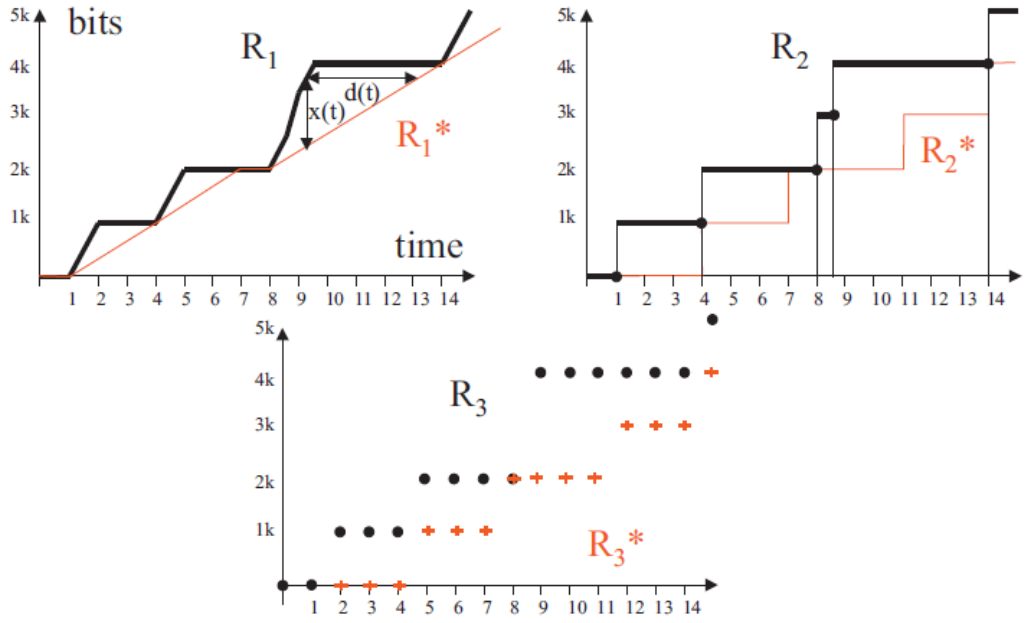


**Figure 3-2 Input and Output Function [53]**

A single server queue is represented in Figure 3-2, both the input function and output function. In this figure, each packet has to wait 3 time units until it been served. The output function $R_1$* which is a fluid model serves the packet when the first bit has been received and the packet departures one bit after another at a fixed rate. For instance, the initial packet arrives during the times 1 to 2 and departures during the times 1 to 4. The output function $R_2$* serves the packet when it has been received completely and is considered leave the system only when this packet has been entirely transmitted. This mechanism is considered as the store and forward mechanism. In $R_2$*, after time 1, the first packet arrives instantly and departures instantly at time 4. In the output function $R_3$*, the first packet arrives at time 2 and leaves at time 5 which is considered as a discrete time model.

The cumulative function R(t) is assumed that it has a derivative $\frac{dR}{dt} = r(t)$ such that $R(t) = \int_0^t r(s)ds$ which is a fluid model. Then the function r is considered as the rate function.

If the time slot δ is chosen and sampled by

$$\boldsymbol{S(n) = R(n\delta)} \qquad \textbf{(3-1)}$$

A continuous time model cumulative function R(t) could be mapped into a discrete time model cumulative function S(n), n∈N.

This mapping could lead to an information loss. A continuous time model cumulative function can be obtained without information loss from S(n), n∈N by letting:

$$\boldsymbol{R'(t) = S(\left\lceil \frac{t}{\delta} \right\rceil)} \qquad \textbf{(3-2)}$$

In this formula, the cumulative function $R'$ is always left-continuous.

By using the formula (3-1), any result obtained from a continuous time model could also be used to the discrete time. In general, the ATM applies the discrete time model while the continuous time model is introduced to dealing with the variable size packets, even not a fluid model.

## 3.2 Virtual Delay

As long as the input and output functions are confirmed, the delay can be obtained.

For a lossless system, the virtual delay at time t is $d(t) = \inf\{\tau \geq 0: R(t) \leq R^*(t+\tau)\}$.

The virtual delay is considered as the horizontal deviation (The deviation of the input and output function in the horizontal (See Figure 3-3). If the input and output function are continuous (fluid model), then it can be seen from the Figure 3-1 that $R^*(t + d(t)) = R(t)$ and that d(t) is the smallest value fitting this formula.

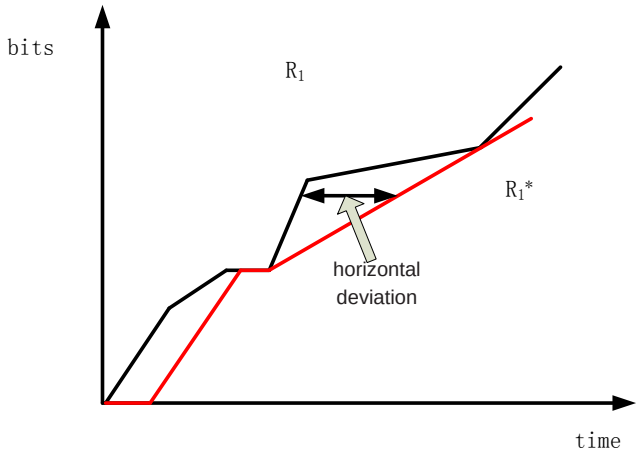


**Figure 3-3 Horizontal Deviation**

In Figure 3-2, the values of virtual delay are different for these three models. In the first subfigure, the delay of the first packet which the last bit experienced is d (2) equals to 2 time units. Moreover, the virtual delay equals to d (1) which is 3 time units on the second subfigure. The virtual delay of the fourth packet on subfigure 2 is d (8.6) equals to 5.4 time units. It takes 2.4 units of waiting time and 3 units of service time to complete the transmission.

## 3.3 Arrival Curves

### 3.3.1 Definition of Arrival Curve

To provide the guarantees to data flows, the traffic sent by the sources should be limited. Arrive curve is proposed to limit this traffic. The concept of arrival curve is defined below.

Define α, which is a wide-sense increasing function. If t ≥ 0 and only if for s ≤ t while the flow R is constrained by α, then

$$R(t)-R(s) \leq \alpha(t-s) \qquad \textbf{(3-3)}$$

In this way, the α is the arrival curve of the function R and the R can be called α-smooth.

As shown in Figure 3-4, the condition is over a set of overlapping intervals.

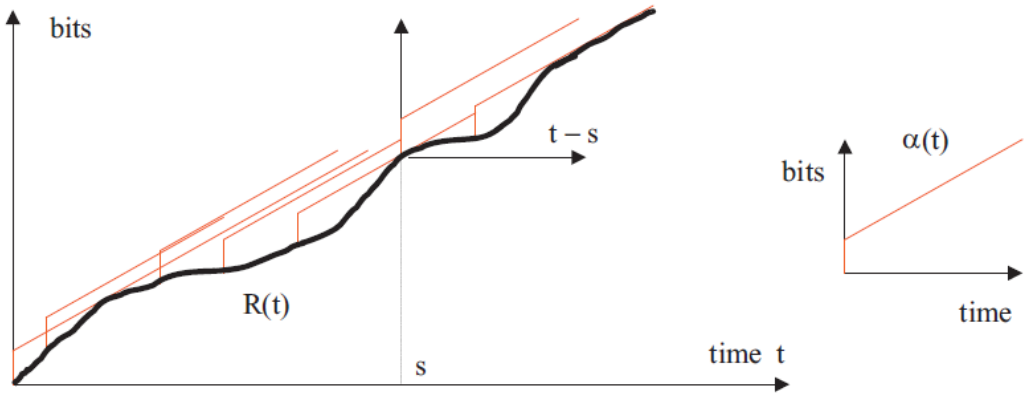


**Figure 3-4 Arrival Curve [53]**

### 3.3.2 Affine Arrival Curves

For instance, if $\alpha(t) = r\tau$, it is to say that, for any time interval $\tau$, the total number of bits of the flow is no more than $r\tau$. It also can be said that this flow has a peak rate. One example is that the fast Ethernet has a restriction rate of 100M/s.

If $\alpha(t) = b$ while b is constant, it means that an arrival curve of the flow will send flow less than b.

According to their relationship with leaky buckets, the affine arrival curves $\gamma_{r,b}$ can be represented below:

$\gamma_{r,b}$ (t) = rt+b for t > 0 and 0 otherwise is proposed.

If $\gamma_{r,b}$ is the arrival curve of one flow, it means that b bits of data will be transferred each time. At the same time, the maximum transmission rate is less than r b/s. In the arrival curve, b represents the burst tolerance and r expresses the rate. Figure 3-4 illustrates these two parameters.

### 3.3.3 Stair Functions as Arrival Curves

The arrival curve is represented as the form $kv_{T,\tau}$ in the ATM. Meanwhile, $v_{T,\tau}$ is used to express the stair function which is listed below:

$V_{T,\tau}(t) = \left\lceil \frac{t+\tau}{T} \right\rceil$ for t > 0 and 0 otherwise.

$v_{T,\tau}(t) = v_{T,0}(t+\tau)$, then $V_{T,\tau}$ can be considered as the results from $v_{T,0}$ by a time shift to the left. In this equation, the time interval is represented as T and the tolerance is expressed as $\tau$. Both parameters are illustrated in time units. Take an ATM flow for example that a flow transfers fixed size packets and up to k unit of data. Assume all packets are separated by the T time units. $Kv_{T,0}$ is the arrival curve of this flow.

If one stream is multiplexed with other streams, it can be considered that the packets of this flow are sent to a queue. At the same time, those flows multiplexed with this flow are also sending data to this same queue. This queue introduces a variable delay which can be considered has a maximum bound equal to τ time units. In this scenario, if R(t) represents the input function while R*(t) is the output function, both functions are for the flow at the multiplexer. As $R^*(s) \leq R(s-\tau)$, it can be deduced that:

$$\boldsymbol{R^*(t) - R^*(s) \leq R(t) - R(s-\tau) \leq Kv_{T,0}(t-s+\tau) = Kv_{T,\tau}(t-s)} \qquad \textbf{(3-4)}$$

Then $Kv_{T,\tau}$ is the arrival curve of R*. It can be inferred that a periodic flow, T represents its period, and k is the fixed size of the packet. If this flow has a variable delay which is less than $\tau$, then $Kv_{T,\tau}$ can be utilised to represent its arrival curve. The function $v_{T,\tau}$ is utilised to demonstrate the minimum time interval between packets.

R(t) is utilised as a left-continuous cumulative function of a flow, while time is discrete or continuous. If this flow sent packets which size are all fixed. Meanwhile, all these packets sizes are k data units and arrival with no time delay. $t_n$ is proposed to represent the arrival time of the nth packet, the properties which illustrated below have the same meaning:

1. for all m, n, $nT - \tau \leq t_{m+n} - t_m$

2. $kv_{T,\tau}$ is the arrival curve of the flow.

Since all the packets sizes are k data units and the packets generate consecutively, it can be inferred that the R(t) can be represented as nk, while n $\in$ N. Moreover, the time interval between two consecutive packets is $\geq T - \tau$.

R(t) is the input function of one flow while the α(t) is assumed as a wide increasing function. For both functions, t ≥ 0. Set $\alpha_l(t)$ the limit of α at t. It can be assumed that $\alpha_l(t) = \sup_{s<t} \alpha(s)$. Moreover, if the function R(t) has an arrival curve which is α, then $\alpha_l$ is also the arrival curve of the function R(t).

R(t) is the input function of one flow, t ∈ R+, or a discrete time flow R(t) while t ∈ N. If this flow sends packets at a fixed size k with no delay, then for some T and τ, set $r = \frac{k}{T}$ and $b = k(\frac{\tau}{T} + 1)$. Then this function R(t) is constrained by $\gamma_{r,b}$ or by $kv_{T,\tau}$.

### 3.3.4 Leaky Bucket and Generic Cell Rate Algorithm

The data of the flow input function R(t) could be analysed by creating a Leaky Bucket Controller. It assumes that the bucket is empty at first and then will contain the fluid which size is b. There is a hole in the bucket and the contents in it would leak at a rate of r units until it is empty.

If the function R(t), known as the amount of data, would not cause the overflow in the bucket, the data is declared conformant. Otherwise, it is non-conformant. The definition of Leaky Bucket is shown in Figure 3-5. The overflow data which can't be poured into the bucket is considered as "non-conformant" data.

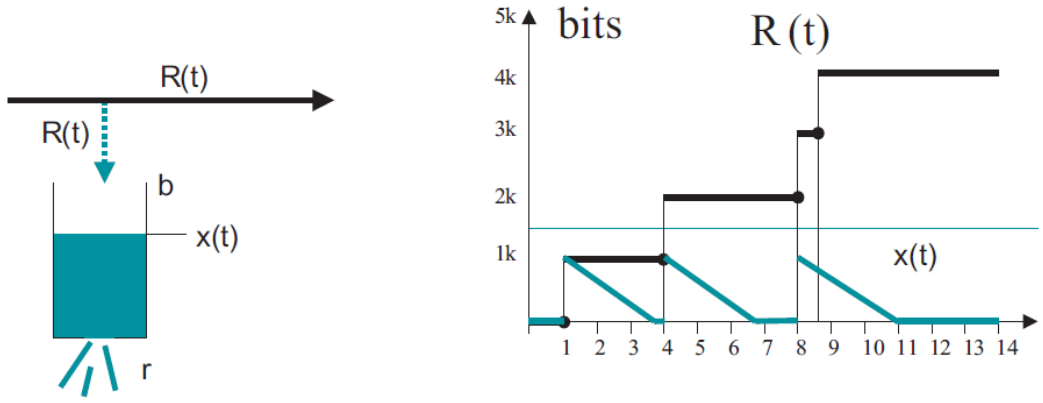


**Figure 3-5 Leaky Bucket Controller [53]**

A buffer with an input function R(t), which the service rate is r. This rate r is set as a fixed rate. In the beginning, there is no data in this buffer. Assumed that all

data received into the buffer during the time interval [0, t], the data in buffer at time t can be illustrated as below:

$$x(t) = sup_{s:s\leq t}\{R(t) - R(s) - r(t-s)\} \quad \textbf{(3-5)}$$

Considering a leaky bucket controller, b is utilised to represent the bucket size. r is proposed to express the leak rate of this controller. It can be said that the arrival curve $\gamma_{r,b}$ constrains the flow of this leaky bucket controller. The following properties are also correct:

1. The flow of conformant data has $\gamma_{r,b}$ as an arrival curve of these conformant data which consist the flow;

2. All data is conformant if $\gamma_{r,b}$ is the arrival curve of the input function.

R(t) is the input function of one flow. Meantime, this flow sends data at fixed size k data units and has no time delay. This input function R(t) can be discrete or continuous. Moreover, the function R(t) is left-continuous. The properties illustrated below have the same meaning:

1. the flow is conformant to GCRA(T, τ)

2. $kv_{T,\tau}$ is an arrival curve of this flow

A flow obeys the GCRA (T, τ) which transfers packets at a fixed size. It is said that this flow also can be represented as the leaky bucket controller. The parameters of this leaky bucket controller r and b can be illustrated as below:

$$b = \left(\frac{\tau}{T} + 1\right)\delta \quad \textbf{(3-6)}$$

$$r = \frac{\delta}{T} \quad \textbf{(3-7)}$$

The packet size is represented by δ in units of data.

Considering a set of I leaky bucket controllers (or GCRAs), $r_i$ expresses the rate of the leaky bucket controller i while the $b_i$ represents the tolerant of the same controller i, for $1 \leq i \leq I$. The flow of the conformant data has an arrival curve as:

$$\alpha(t) = min_{1\leq i\leq I}\left(\gamma_{r_i,b_i}(t)\right) = min_{1\leq i\leq I}(r_i t + b_i) \quad \textbf{(3-8)}$$

## 3.4 Service Curves

As mentioned above, the arrival curve is proposed to constrain the flows of the integrated services networks. Guarantees to flows are needed for reservations purposes which are done by packet schedulers [54]. The service curve is introduced to abstract the details of the packet scheduling. Two examples are presented to get a better understanding of the service curve.

The first example of a scheduler is a Generalized Processor Sharing (GPS) node [55]. A GPS node serves several flows simultaneously. Each flow is set at a given rate. Moreover, the node has some backlogs as well. The guarantee is that during a time window t, if the given rate of the node is r, thus it receives data which amount is, at least, equal to rt. Since the GPS node bases on a fluid model, it only can be considered as a theoretical concept. In the real network, node transfers with packets.

Consider a GPS node with an input function R and output function $R^*$. The given rate is r. In this scenario, assumed overflow will not occur. For all time t, considered $t_0$ as the beginning of the last busy period for the flow up to time t. It can be obtained that:

$$\boldsymbol{R^*(t) - R^*(t_0) \geq r(t - t_0)} \qquad \textbf{(3-9)}$$

In this part, the function R is assumed as left-continuous. At time $t_0$ there is no data which means backlog equals to 0 as $R(t_0) - R^*(t_0) = 0$. Combining this formula with the formula (3-9):

$$\boldsymbol{R^*(t) - R(t_0) \geq r(t - t_0)} \qquad \textbf{(3-10)}$$

Then it can be shown that, for all time t: $R^*(t) \geq \inf_{0 \leq s \leq t} [R(s) + r(t - s)]$, which can be written as

$$\boldsymbol{R^* \geq R \otimes \gamma_{r,0}} \qquad \textbf{(3-11)}$$

In the formula (3-11), the operational character $\otimes$ represents the min-plus convolution. F is a wide-sense increasing function equals to zero when negative

arguments. If f and g are two curves of F, the min-plus convolution can be represented as $(f \otimes g)(t) = \inf_{0 \le u \le t}(f(t-u) + g(u))$.

Here is the second example. Assume a network node which the maximum delay of a given flow R is bounded by T. The first in, first out mechanism is proposed in this example. The delay bound could be translated to d(t) ≤ T for all t. Since $R^*$ is always wide-sense increasing, according to the definition of d(t), it can be obtained that $R^*$ (t + T) ≥ R(t). On the contrary, it can be deduced that if $R^*$ (t + T) ≥ R(t), then d(t) ≤ T. It can be said that if $R^*$ (t + T) ≥ R(t) for all t, the maximum delay is bounded by T. This formula can also be expressed as

$$\boldsymbol{R^*(s) \ge R(s-T)} \qquad \textbf{(3-12)}$$

for all s ≥ T.

The "impulse" function $\delta T$ defined by

$\delta_T(t) = 0$ if $0 \le t \le T$ and $\delta_T(t) = +\infty$ if $t > T$.

For any wide-sense increasing function x(t), defined for $t \le 0$, $(X \otimes \delta T)(t) = X(t-T)$ if t ≥ T and $(X \otimes \delta T)(t) = X(0)$ otherwise. Then the maximum delay can thus be illustrated as

$$\boldsymbol{R^* \ge R \otimes \delta T} \qquad \textbf{(3-13)}$$

It can be seen from these two examples above that an input-output relationship of the same form is inferred (formula (3-12) and (3-13)).

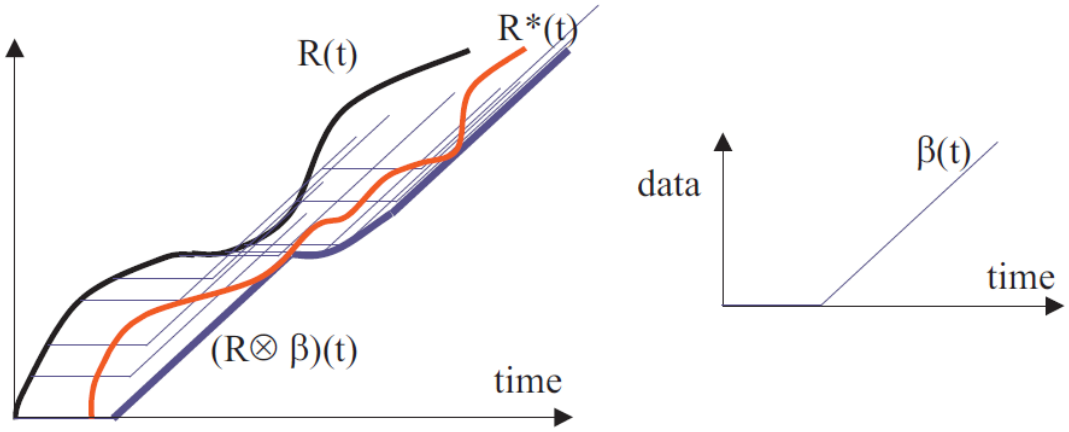


**Figure 3-6 Service Curve [53]**

Consider a system S with a flow through it. This flow has the input and output function which are R and $R^*$. It can be said that system S offers to the flow a service curve β if and only if β is wide sense increasing. At the same time, β(0) = 0 and $R^* \geq R \otimes \beta$.

If the service curve β is continuous, it means that for all t, when $t_0 \leq t$

$$\boldsymbol{R^*(t) \geq R_l(t_0) + \beta(t - t_0)} \qquad \textbf{(3-14)}$$

Where $R_l(t_0) = sup_{\{s<t_0\}} R(s)$ is the limit to the left of R at $t_0$. If R is left-continuous, then $R_l(t_0) = R(t_0)$.

## 3.5 Network Delay Bound

In the network calculus, one important quantity is the maximum horizontal deviation between two curves f and g of the function F. The function F is a wide-sense increasing function equals to zero when negative arguments. This horizontal deviation is represented as h(f,g) while $h(f,g) = \sup_{t\geq 0}\{\inf\{d \geq 0 | f(t) \leq g(t+d)\}\}$

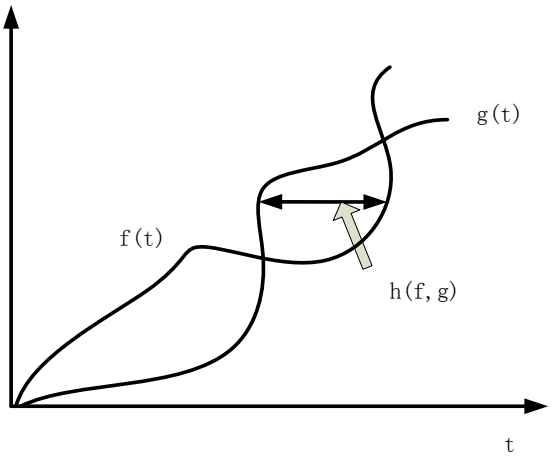


**Figure 3-7 The Horizontal Deviation Between f and g**

Assume a flow, constrained by the arrival curve α, traverses a system that offers a service curve of β. The virtual delay d(t) for all t satisfies: d(t) ≤ h(α,β). Then h(α,β) is the network delay bound this flow experiences.

## 3.6 AFDX Network Time Delay Analysis

### 3.6.1 AFDX Network Time Delay Model

Three delays which consist of the AFDX network delay are the end system delay, the AFDX switch delay and the propagation delay (See Figure 3-8).

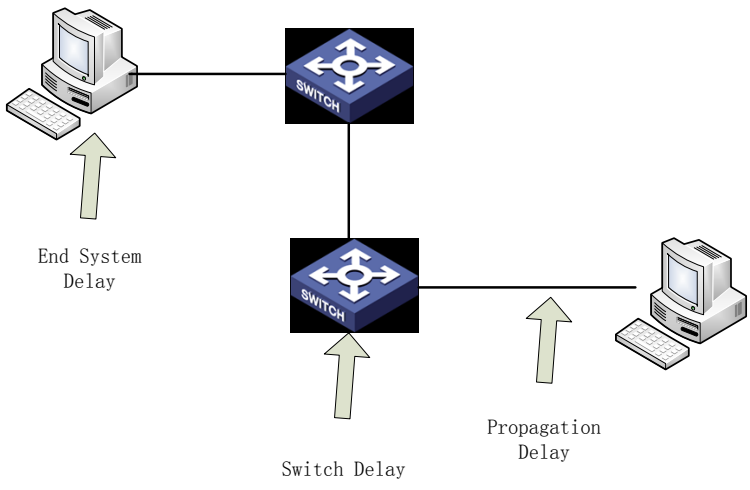


**Figure 3-8 AFDX Network Time Delay**

The end system delay is consisted of the Band Allocated Gap, the scheduler delay and the hardware process delay.

The switch delay is consisted of the scheduler delay and hardware process delay.

The propagation delay is consisted of the frames transmission delay and the signal transmission delay.

### 3.6.1.1 End System Time Delay Model

The end system is used to send and to receive the AFDX packets between the avionics applications and the AFDX switches. All these packets are generated by the avionics applications and transmitted through the virtual links. The traffic regulator introduces the BAG to control the traffic rate on each virtual link which constraints the bandwidth utility of each virtual link. The scheduler handles the concurrency data traffic depending on the particular scheduling mechanism which introduces the additional latency (See Figure 3-9).

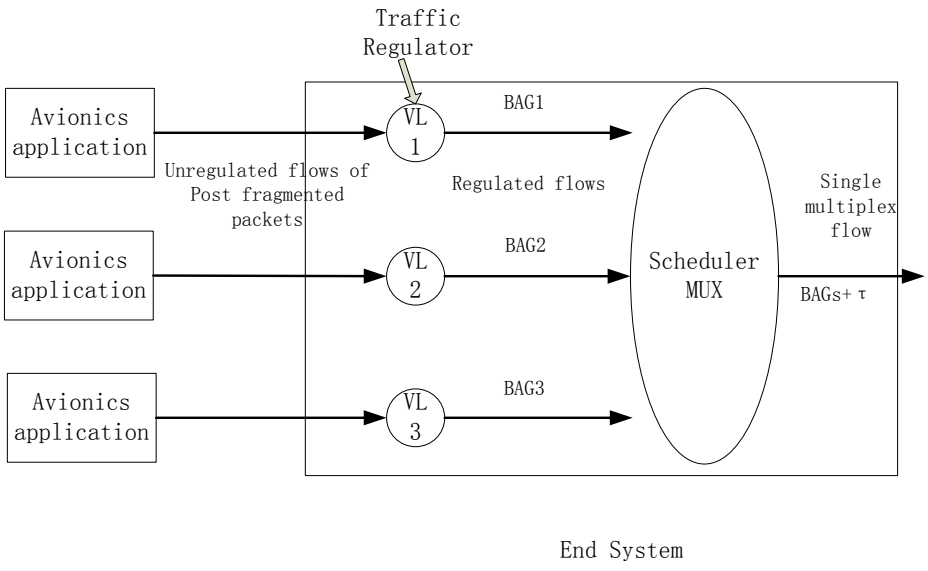


**Figure 3-9 End System Delay Model**

### 3.6.1.2 Switch Model

An AFDX switch is proposed to receiving the AFDX packets from the source end system and forwarding those packets to the destination end systems.

Usually, the AFDX switch contains one FIFO buffer for each output port. Also, the input ports buffers are not involved. This is common in switch models [14][15][56] .

An example of the AFDX switch model for virtual link scheduling is shown in Figure 3-10.

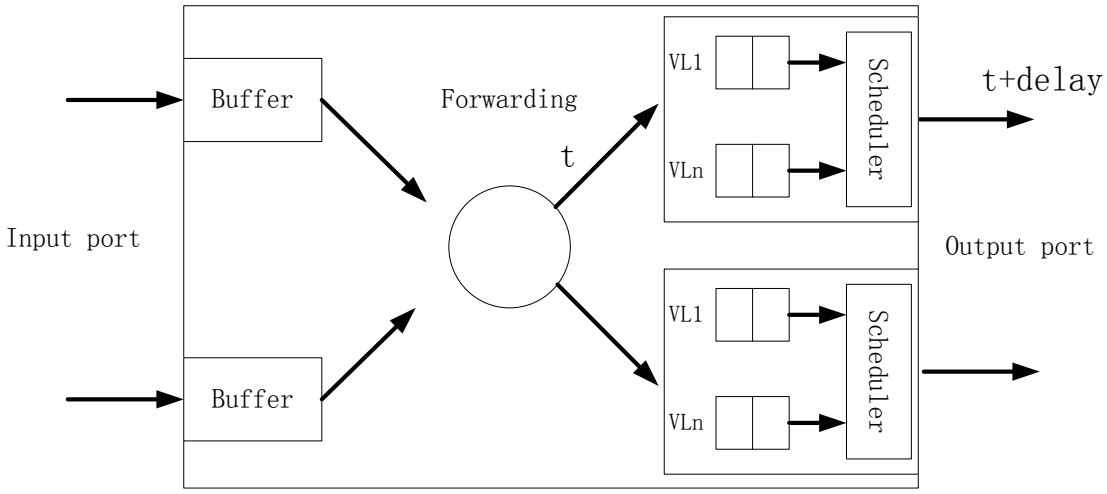


**Figure 3-10 AFDX Switch Time Delay Model**

The Ethernet frames transmit through the exclusive buffers of input ports. It is known that the minimum size of the frame is 576 bits while the minimum inter-frame gap is 96 bit times. When the 10 Mb/s Ethernet is proposed, the minimum transmission time is 57.6+9.6 microseconds. If this frame transmits on 100Mb/s Ethernet, the transmission time is 5.76+0.96 microseconds.

This switch module forwards the received packet from the input port to destination virtual link queue by checking a static forwarding table. This check needs a stationary time cost no matter how large the packet size is. To meet this requirement, a powerful forwarding capacity of AFDX switch should possess which can transfer each received packet to its corresponding output port queue before the next packet arrives.

As shown in Figure 3-10, the structure of multiple virtual links queues and a scheduler consolidated at each output port are familiar to the end system. In this section, the output waiting in the queue is the main delay component which

will be analysed. Messages at each output port in the virtual link queues will be scheduled and then transferred into the physical link.

### 3.6.2 AFDX Traffic and Service Model

In this part, a brief introduction to the theory of network calculus will be given, and then the approaches utilising network calculus to represent the AFDX data flows and the entities will then be illustrated.

According to [53], the network data flows can be modelled by the leaky bucket or generic cell rate algorithm, i.e. the (σ,ρ) model and the GCRA model. The rate-latency function $\beta(t) = R \cdot (t - T)$ is the general way to illustrate the service curve of AFDX network. In this formula, R is the service rate while T is the sum of AFDX node initial time and the queuing delay [57] [58].

#### 3.6.2.1 AFDX Illustration with Network Calculus

The AFDX virtual link is usually described by the leaky bucket model which illustrates the virtual link flow after being regulated by the BAG and before this flow enters the end system scheduler. The arrival curve of this model can be represented as below:

$$\boldsymbol{\alpha_0(t) = \frac{L_{max}}{BAG} t + L_{max}} \qquad \textbf{(3-15)}$$

Where:

$L_{max}$ represents the largest frame size of this virtual link.

$BAG$ expresses the minimum time interval between two consecutive frames of the same virtual link.

In this project, the traffic regulator is assumed to be powerful enough which can process and transfer frames instantly. No extra delay is imposed. Thus, the delay from traffic regulator will be added to the end-to-end delay for obtaining the total time delay.

The AFDX data flow leaves the end system but before enters the first switch is modelled and represented in Figure 3-11. Since there is only one end system scheduler for various virtual links, it also assumes that the scheduler in end

system brings a jitter τ. Then the (σ,ρ) and GCRA model can be expressed as follows.

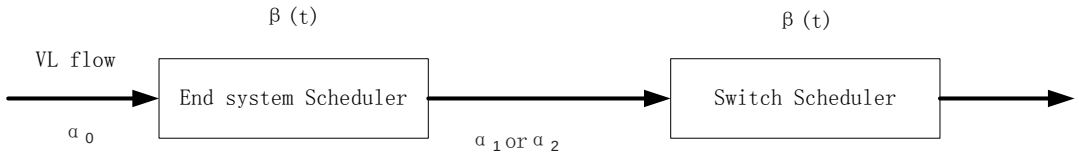


**Figure 3-11 Virtual Link Model Cross End System and Switch**

According to the (σ,ρ) model, the virtual link coming out from the end system is modelled as

$$\boldsymbol{\alpha_1(t) = \frac{L_{max}}{BAG} t + L_{max}\,(1 + \frac{\tau}{BAG})} \qquad \textbf{(3-16)}$$

Where:

$L_{max}$ is as same as represent by formula (3-15).

$BAG$ is as same as represent by formula (3-15).

$\tau$ is jitter imposed by the scheduler in the end system.

According to [53], τ can be measured but should be excluded from the end-to-end delay. Therefore, the sustainable rate of flow σ is $L_{max} \ / \ BAG$, and burst ρ is $L_{max} \ . (1 + \frac{\tau}{BAG})$.

When the GCRA model is applied, the virtual link flow can be illustrated as

$$\boldsymbol{\alpha_2(t) = L_{max} \left\lceil \frac{(t+\tau)}{BAG} \right\rceil} \qquad \textbf{(3-17)}$$

Where:

$L_{max}$ is as same as represent by formula (3-16).

$BAG$ is as same as represent by formula (3-16).

$\tau$ is as same as represent by formula (3-16).

The periodic tasks usually utilise the GCRA model to illustrate their flow characters, especially the network flows.

The rate-latency service curve proposed to illustrate the service capability of the scheduler of the switch and end system is listed below:

$$\boldsymbol{\beta(t) = R(t - T_0 - T_s)} \qquad \textbf{(3-18)}$$

Where:

$R$ is the service rate of the output port of end system or switch, for instance, 10/100 Mbps in AFDX network.

$T_0$ represents the initial time of system.

$T_s$ expresses the queuing delay due to the resource competition among the virtual links.

As mentioned before, the horizontal distance between the arrival curve and service curve is the end-to-end delay bound. Basing on the “Pay Burst Only Once” phenomenon, once a data flow traverses several service nodes, the sequential service nodes can be treated as one integrated node. For this integrated node, its service curve is the convolution of the service of all the nodes which have been consolidated into this integrated node.

### 3.6.2.2 End-to-End Delay Analysis

As can be seen in Figure 3-12, a single flow passes the switch scheduler is shown. In this section, the latency of this flow will be analysed, both GCRA model and (σ,ρ) model. Since this model is as same as the general process sharing model, the queuing delay $T_s$ in formula (3-18) is set to 0.

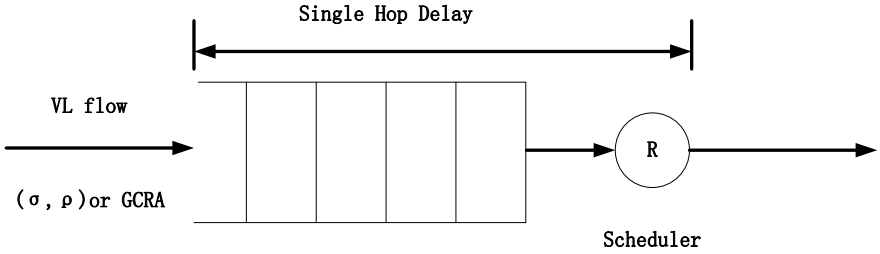


**Figure 3-12 Delay Bound For Single Virtual Link Cross Scheduler**

(Delay bound for (σ,ρ) model) An AFDX node S has a rate latency service curve β. As defined in the formula (3-18), the service curve $\beta(t) = R(t - T_0 - T_s)$. Since the $T_s$ equals to 0, the service curve is $\beta(t) = R(t - T_0)$. Assume that the arrival curve of the input flow follows the (σ,ρ) model. Thus, it can be assumed that the delay bound of the flow traverses the node is

$$\boldsymbol{D_{(\sigma,\rho)} = T_0 + \frac{L_{max}\left(1+\frac{\tau}{BAG}\right)}{R}} \qquad \textbf{(3-19)}$$

Where:

$L_{max}$ is the maximum frame size.

$T_0$ is the initial time of the node.

$\tau$ is the jitter .

$R$ is the network service rate.

This formula has been proved from the Theorem 1.4.3 [53].

(Delay Bound for GCRA Model) An AFDX node S has a rate latency service curve β. As defined by the formula (3-18), the service curve $\beta(t) = R(t - T_0 - T_s)$. Since $T_s$ = 0, service curve is $\beta(t) = R(t - T_0)$. Assume that the arrival curve of the input flow follows the GCRA model. Moreover, the service curve of the rate is greater than the arrival curve. Then it can be inferred that the delay bound for the data flow is

$$\boldsymbol{D_{GCRA} = \frac{L_{max}}{R} + T_0 + \left[\frac{L_{max}}{R} - (BAG - \tau)\right]^+} \qquad \textbf{(3-20)}$$

Where:

$L_{max}$ is the packet size.

$R$ is the network service rate.

$T_0$ is the network node initial time.

$BAG$ is the period of flow.

$\tau$ is the release jitter.

Note that $[X]^+$ means max (0, x).

Proof: The delay bound of the first packet in the flow can be illustrated as below

$$\boldsymbol{D_1 = \frac{L_{max}}{R} + T_0} \quad \textbf{(3-21)}$$

The delay bound of the second packet in the flow can be expressed as below

$$\boldsymbol{D_2 = \frac{2L_{max}}{R} + T_0 - (BAG - \tau)} \quad \textbf{(3-22)}$$

According to [53], the T-SPEC model can be applied to the arrival curve of the AFDX GCRA model with α(t) = min(M + pt, rt + b) . Furthermore, the service rate R is larger than the rate of arrival curve r which infers that the delay of the second packet is bigger than the succeeding packets. In this way, the upper bounds delay will occur when receiving the first two packets. By utilising the formula (3-21) and (3-22), the delay bound for the flow can be illustrated:

$$\boldsymbol{D_{GCRA} = \frac{L_{max}}{R} + T_0 + \left[\frac{L_{max}}{R} - (BAG - \tau)\right]^+} \quad \textbf{(3-23)}$$

The GCRA model provides a tighter delay bound than the (σ,ρ) model at a heavy load network.

Proof: According to formula (3-19) and (3-23), the subtraction of delay bound obtained by the (σ,ρ) model and the GCRA model is represented below

$$\boldsymbol{D_{(\sigma,\rho)} - D_{GCRA} = \frac{L_{max}\cdot\tau}{R\cdot BAG} - \left[\frac{L_{max}}{R} - (BAG - \tau)\right]^+} \quad \textbf{(3-24)}$$

If $\frac{L_{max}}{R} - (BAG - \tau) \leq 0$, then it can be inferred that $D_{(\sigma,\rho)} - D_{GCRA} = \frac{L_{max}}{R} \cdot \frac{\tau}{BAG} > 0$. On the contrary, if $\frac{L_{max}}{R} - (BAG - \tau) > 0$, then $D_{(\sigma,\rho)} - D_{GCRA} = \frac{L_{max}}{R} \cdot \frac{\tau}{BAG} - \frac{L_{max}}{R} + BAG - \tau = (BAG - \tau)\left(1 - \frac{L_{max}}{R\cdot BAG}\right)$. Since the service rate (R) is always larger than the arrival rate $\left(\frac{L_{max}}{BAG}\right)$, It can be deduced that $\frac{L_{max}}{R\cdot BAG} < 1$, $\left(1 - \frac{L_{max}}{R\cdot BAG}\right) > 0$, then $D_{(\sigma,\rho)} - D_{GCRA} > 0$.

### 3.6.3 End-to-End Delay of GCRA Model

In this section, the service curve offered by only one switch scheduler to each virtual link flow will be analysed firstly, and the end-to-end delay of one particular virtual link flow across this switch scheduler will be gained. Secondly, the total time delay of a given virtual link flow which across several nodes will be obtained. Moreover, the phenomenon “Pay Burst Only Once” is considered.

#### 3.6.3.1 Single Hop Delay Bound Analysis

As can be seen from Figure 3-13, a scheduler is applied to distribute the network resource among virtual links. The service curve provided to each virtual link flow and the corresponding delay bound would be obtained with the following content.

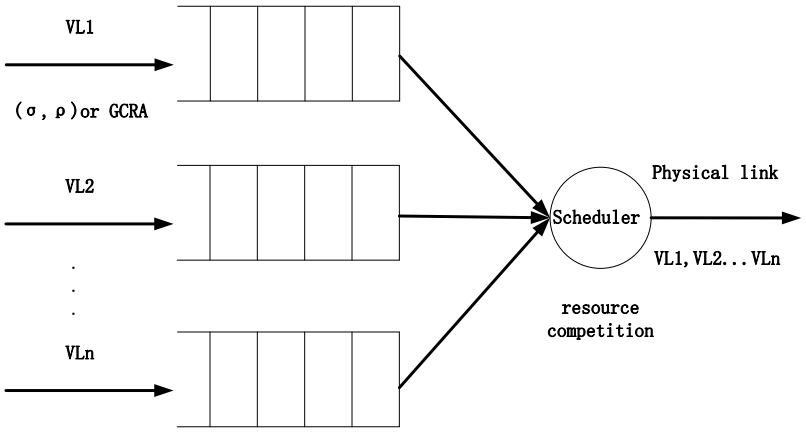


**Figure 3-13 Switch Scheduling [39]**

Assuming that an AFDX Switch scheduler M offers a rate latency service curve β to multiple virtual links flows as defined by the formula (3-18). Then the service curve provided to each single flow j can be represented as below

$$\boldsymbol{\beta_j(t) = R \cdot \left( t - T_0 - \frac{\sum_{s \in S}^{s \neq j} L_{s\,max}}{R} \right)} \qquad \textbf{(3-25)}$$

Where:

$S$ represents the set of all incoming flows.

$L_{s\,max}$ is the maximum frame size of flow s.

If all the arrival curves follow the (σ,ρ) model which can be illustrated as

$$\alpha_{j1}(t) = \frac{L_{j\,max}}{BAG_j} t + L_{j\,max}\left(1 + \frac{\tau_j}{BAG_j}\right) \quad \textbf{(3-26)}$$

Then the delay bound can be deduced as follow

$$D_{j\_(\sigma,\rho)} = T_0 + \frac{\sum_{s\in S} L_{s\,max} + L_{j\,max}\frac{\tau_j}{BAG_j}}{R} \quad \textbf{(3-27)}$$

If all the arrival curves follow the GCRA model which can be illustrated as

$$\alpha_{j2}(t) = L_{j\,max}\left\lceil \frac{t+\tau_j}{BAG_j} \right\rceil \quad \textbf{(3-28)}$$

Then the delay bound is

$$D_{j\_GCRA} = \frac{L_{j\,max}}{R} + T_0 + \frac{\sum_{s\in S}^{s\neq j} L_{s\,max}}{R} + \left[\frac{L_{j\,max}}{R} - (BAG_j - \tau_j)\right]^+ \quad \textbf{(3-29)}$$

Proof: When a packet in flow j arrives at the scheduler, it may be blocked by only one packet of other flow since the forwarding processor has been assumed powerful enough to forward the first packet before the succeeding packet arrives. Therefore, the worst case queuing time is $\sum_{s\in S}^{s\neq j} \frac{L_{s\,max}}{R}$ as a result of the non-preemption of the presently processed packets. Then the scheduler will process the packet with full service rate R which makes $\beta_j(t) = R \cdot (t - T_0 - \sum_{s\in S}^{s\neq j} \frac{L_{s\,max}}{R})$ to be the service curve offered to flow i.

### 3.6.3.2 Multi-Hop End-to-End Delay Analysis

The packet end-to-end time delay consists of three main parts which are the delay in the end system, delay in the switch and delay on the link propagation. This end-to-end delay can be illustrated below:

$$Delay = D_{ES} + D_{Switch} + D_{propagation} \quad \textbf{(3-30)}$$

Where:

$D_{ES}$ is the latency spent in end system.

$D_{Switch}$ is the latency expended in switch.

$D_{propagation}$ is the propagation delay of the link propagation.

The propagation delays on link propagation are assumed upper bounded by $T_{prop}^{max}$.

The total service curve of a given VL flow should be obtained first to get the multi-hop end-to-end delay bound. After that, the total service curve should be applied to the formula (3-29).

### 3.6.3.3 Multi-Hop Total Service Curve

In [58], the end-to-end service curve for a given flow is represented as below:

$$\boldsymbol{\beta_{e2e} = min_{h\in 1,\dots,H}\, R_h(t - H\frac{L_{max}}{R_{trans}} - HT_f - \sum T_{kh})} \qquad \textbf{(3-31)}$$

Where:

$H$ is the total number of the switch nodes.

$R_h$ is the service rate provided to the given flow at hop h.

$Min(R_h)$ is the minimum service rate of all network nodes through the transmission path for a given flow.

$R_{trans}$ is the physical transmission rate.

$T_f$ denotes the time in checking the forwarding table.

$T_{kh}$ is the queuing time in node h.

By applying the formula (3-29) to gain a given virtual link flow when the switch applies Round-Robin as its schedule policy, the total service curve for given flow $VL_j$ can be expressed by the following formula.

$$\boldsymbol{\beta_{j\ e2e(RR)} = min_{1\le h\le H}\left(R_{hj}\right)\cdot\left(t - H\cdot\frac{L_{jmax}}{R_{trans}} - H\cdot T_f - H\cdot T_0 - \sum_{1\le h\le H}\frac{\sum_{s\in S_h\ and\ s\neq j} L_{smax}}{R_{hj}}\right)}$$

**(3-32)**

Where:

$S_h$ is the set of flows coming into the node h.

$R_{hj}$ is the service rate provided to the flow j in node h.

### 3.6.3.4 Multi-Hop End-to-End Delay Analysis

In this section, a given virtual link is assumed to traverse through H switches. By computing the maximum horizontal distance between the virtual link arrival curve and total service curve, after adding the link propagation end-to-end delay, the end-to-end delay is obtained.

When switches use the scheduler and each virtual link follows the traditional (σ,ρ) model, the bound for delay without BAG of flow j is illustrated as below

$$\boldsymbol{D_{j\,e2e(\sigma,\rho)} = \frac{L_{j\,max}(1+\frac{\tau_j}{BAG_j})}{min_{1\le h\le H}(R_{hj})} + H\cdot\frac{L_{j\,max}}{R_{trans}} + H\cdot T_f + H\cdot T_0 + \sum_{1\le h\le H}\frac{\sum_{s\in S_h}^{s\neq j} L_{s\,max}}{R_{hj}} + (H+1)T_{max}^{prop}} \quad \textbf{(3-33)}$$

Proof: The delay in end system is $\frac{L_{j\,max}\,(1+\frac{\tau_j}{BAG_j})}{\min_{1\le h\le H}(R_{hj})}$, the propagation delay is $(H+1)T_{max}^{prop}$, and the delay in all H switches is $H\cdot\frac{L_{j\,max}}{R_{trans}} + H\cdot T_f + H\cdot T_0 + \sum_{1\le h\le H}\frac{\sum_{s\in S_h}^{s\neq j} L_{s\,max}}{R_{hj}}$

Hence combining (3-27) and (3-31), formula (3-33) is obtained.

When switches utilises the scheduler and each virtual link follows the GCRA model, the delay bound without BAG is illustrated as below

$$\boldsymbol{D_{j\,e2e(GCRA)} = \frac{L_{j\,max}}{min_{1\le h\le H}(R_{hj})} + H\cdot\frac{L_{j\,max}}{R} + H\cdot T_f + H\cdot T_0 + \sum_{1\le h\le H}\frac{\sum_{s\in S_h}^{s\neq j} L_{smax}}{R_{hj}} + [\frac{L_{j\,max}}{min_{1\le h\le H}(R_{hj})} - (BAG_j - \tau_j)]^+ + (H+1)\cdot T_{max}^{prop}} \quad \textbf{(3-34)}$$

Proof: The proof is similar to the formula (3-33) which will not be listed again.

## 3.7 Total Time Delay Analysis

As elaborated above, the delay among the end system scheduler, switch and transmission propagation has been achieved, the (σ,ρ) model and GCRA model respectively. This project tries to obtain the total delay from the traffic regulator, end system scheduler, switch and transmission propagation. As assumed before, the traffic regulator in this project is powerful enough which will not

impose an extra delay to the whole communication. Thus, the entire transmission time this project focuses on can be illustrated as below:

For the (σ,ρ) model

$$\boldsymbol{D_{j\,L_{max}(\sigma,\rho)} = BAG_j + \frac{L_{j\,max}(1+\frac{\tau_j}{BAG_j})}{min_{1\le h\le H}(R_{hj})} + H\cdot\frac{L_{j\,max}}{R_{trans}} + H\cdot T_f + H\cdot T_0 + \sum_{1\le h\le H}\frac{\sum_{s\in S_h}^{s\neq j} L_{s\,max}}{R_{hj}} + (H+1)T_{max}^{prop}} \quad \textbf{(3-35)}$$

For the GCRA model

$$\boldsymbol{D_{j\,L_{max}(GCRA)} = BAG_j + \frac{L_{j\,max}}{min_{1\le h\le H}(R_{hj})} + H\cdot\frac{L_{j\,max}}{R} + H\cdot T_f + H\cdot T_0 + \sum_{1\le h\le H}\frac{\sum_{s\in S_h}^{s\neq j} L_{smax}}{R_{hj}} + [\frac{L_{j\,max}}{min_{1\le h\le H}(R_{hj})} - (BAG_j - \tau_j)]^+ + (H+1)\cdot T_{max}^{prop}} \quad \textbf{(3-36)}$$

According to [39], the GCRA model could obtain a tighter delay bound than the (σ,ρ) model. To obtain a tighter delay, the GCRA model is proposed in this project.

During the AFDX data divided process, each divided message will be added the extra data which are the source port, destination port, destination IP address, sequencer number and length of data. All those data occupy another 4+4+4+4+4=20 bytes. This makes each divided message only contains ($L_{max}$-20) bytes of AFDX data. Thus, the total delay of the whole AFDX network can be illustrated as

$$\boldsymbol{D_{j\,whole\,time(GCRA)} = \left\lceil\frac{P}{(L_{j\,max}-20)}\right\rceil\cdot(BAG_j + \frac{L_{j\,max}}{min_{1\le h\le H}(R_{hj})} + H\cdot\frac{L_{j\,max}}{R} + H\cdot T_f + H\cdot T_0 + \sum_{1\le h\le H}\frac{\sum_{s\in S_h}^{s\neq j} L_{smax}}{R_{hj}} + [\frac{L_{j\,max}}{min_{1\le h\le H}(R_{hj})} - (BAG_j - \tau_j)]^+ + (H+1)\cdot T_{max}^{prop})} \quad \textbf{(3-37)}$$

In this formula, P is the data size sent by the avionics application.

In each experiment, the avionics data has a fixed size 640 bytes. In this project, the $T_f$ is set to 10 μs according to the datasheets of switches used in this simulation. $T_0$ is 0 due to the mechanism of the FACADE platform. Meantime, τ is also considered as 0 since no more than three virtual links will execute

simultaneously. That is to say, the total time delay values calculated in this thesis are all acquired by using the GCRA model with formula (3-37).

## 3.8 Summary

This chapter introduces the network calculus. Firstly, the definitions of the cumulative function, arrival curve as well as service curve are introduced. Then the delay bound is represented which can be deduced by the arrival curve and service curve. Secondly, the end system delay model and the AFDX switch delay model are represented. Thirdly, the arrival curve and service curve of AFDX network are illustrated, the (σ,ρ) model and GCRA model respectively. Fourthly, the latency for a single flow cross the switch scheduler is analysed. Fifthly, the end-to-end delay is illustrated, from the single hop delay to the multiple hop delay, both the (σ,ρ) model and GCRA model. Finally, the total time delay of this project bases on the GCRA model is obtained.

# 4 DEVELOPMENT OF FACADE AND AVIONICS APPICATION SIMULATION PLATFORM

## 4.1 The Framework of FACADE

The AFDX network simulation platform applied in this thesis named FACADE is same as the previous project [59] [60] [61]. Some modifications have been made to obtain an appropriate platform for experiment purpose. The architecture of the FACADE is shown in Figure 4-1.

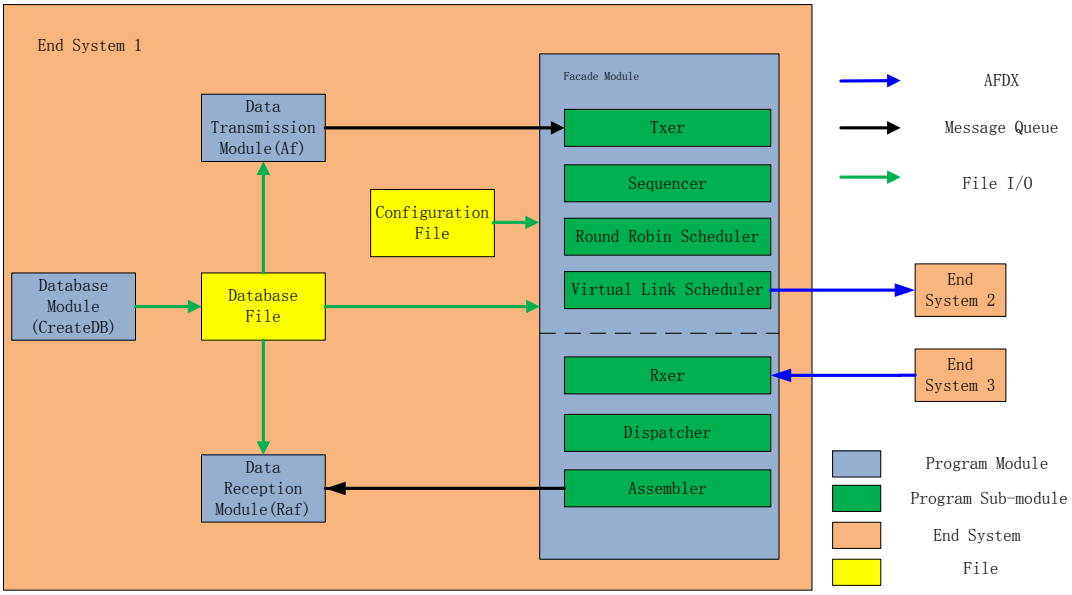


**Figure 4-1 Architecture of FACADE**

As presented in the figure, there are four main modules in the FACADE platform which are the Database module, FACADE module, Data Transmission module and Data Reception module. The main functions of each module will be shown below:

### Database Module

The database files can be built by two different methods, creating and maintaining by users manually or generating by platform automatically. The manual mode could increase the complex of the platform utilisation and may cause the abnormal of the platform if the database file goes wrong. In this platform, the database files are built by the platform itself. The Database

module creates database files for each end system respectively. Those database files contain the parameters of the source end systems, destination end systems as well as virtual links. All those parameters should be loaded when other modules are executing. In other words, the Database module should be executed before other modules to create database files. When the Database module is executed, the parameters stored in database files will be printed on the screen for error correction.

**Data Transmission Module (Af)**

The Data Transmission module is proposed to receive the AFDX data from the avionics application. It acts as an interface of the FACADE platform. After receiving the data, the Data Transmission module will transfer those data to the Txer sub module.

**Data Reception Module (Raf)**

The Data Reception module is proposed to receive the AFDX data from the Assembler sub module. It acts as an interface of the FACADE platform. After receiving the data, the Data reception module transfers those data to the avionics application, the transmission completed.

**FACADE Module**

The FACADE module is the main part of the whole FACADE platform. It can be divided into two main partitions: the AFDX transmission partition and the AFDX reception partition.

The AFDX transmission partition contains four sub modules: Txer, Sequencer, Round Robin Scheduler and Virtual Link Scheduler. The Txer sub module is invited to initialize each related sub module running environment and to transmit the AFDX data. After initialization, the Txer sub module will receive the AFDX data from the Af module. After that, those received AFDX data will be transformed into the AFDX fragmented messages. Then those fragmented messages will be transferred to corresponding destination end systems thought the UDP port by using UDP socket.

The initialization of Txer sub module contains several functions which are:

- Loading the configuration by reading the configuration files.
- Loading the parameters of the end systems and virtual links by reading the database files.
- Initialize the message queue, semaphore and Mutex for corresponding sub modules.
- Allocating the storage memory for the running environment parameters.
- Executing and managing the other modules of the AFDX transmission partition.

The function of Sequencer, Round Robin Scheduler and Virtual link Scheduler are listed below:

- ***Sequencer:*** This sub module divided received AFDX data into fragmented messages. After that, the fragmented messages will be added a sequence number and then transmitted into corresponding sub-virtual links.
- ***Round Robin Scheduler:*** This sub module is proposed to transmit the messages from sub-virtual link to corresponding virtual link follow the round robin algorithm.
- ***Virtual Link Scheduler:*** This sub module sends fragmented messages over the virtual link to corresponding destination end system through the UDP port by utilising UDP socket. The parameters of the virtual link are stored in the database files which have been loaded by the Txer sub module.

Another partition, AFDX reception partition, is used to receive the AFDX fragmented messages and convert them into the original AFDX data. Then those data will be sent to the Raf module. The AFDX reception partition contains three sub modules: Rxer, Dispatcher and Assembler.

The Rxer sub module listens to the specific UDP ports which have been predefined in the database files. Those AFDX fragmented messages sent by the source end systems will then be received through the UDP port by using UDP socket. The contents and sequence numbers of those messages will be

obtained by reading those messages. After that, those messages are assembled and retrieved into the original data. Finally, those data will be sent to the Raf module.

The Rxer sub module is woken by the Txer sub module. It also has to initialize the running environment for corresponding modules. The initialization functions are:

- Loading the configuration by reading the configuration files.
- Loading the parameters of the end systems and virtual links by reading the database files.
- Initializing the message queue, semaphore and Mutex for corresponding sub modules.
- Allocating storage the memory for the running environment parameters.
- Executing and managing other sub modules of the AFDX reception partition.

The function of Dispatcher and Assembler are listed below:

- ***Dispatcher:*** Receiving the fragmented messages from the Rxer sub module, obtaining the contents and sequencer number of each message and send them to the Assembler sub module.
- ***Assembler:*** Receiving the fragmented messages from Dispatcher sub module, assembling those messages by orders and retrieve them into the original AFDX data. After that, all these original AFDX data will be transferred to the Raf module.

## 4.2 Data Exchange Behaviour

As represented in the Figure 4-2, after every module and sub module are ready, the avionics application can start to transmit the AFDX data at any time. When AFDX data have been sent by the avionics application, those data will be received by the Af module first. Secondly, the AFDX data will be sent to the Txer sub module. The Txer sub module will obtain the contents of the data and then sent them to the Sequencer sub module. The Sequencer sub module is invited to divide AFDX data packets into the fragmented messages. Then those consecutive fragmented messages will be added a sequence number

sequentially and sent to the corresponding sub-virtual links. After that, those fragmented messages will be managed by the Round Robin Scheduler sub module. This module is proposed to transmit those fragmented messages from the sub-virtual links to corresponding virtual links following the round robin algorithm. After the fragmented messages reaching their corresponding virtual links, they will be sent to the corresponding destination end systems via specific UDP port which is defined in the database files by using the UDP socket.

After been sent by the Virtual Link Scheduler sub module, those fragmented messages will be transmitted to the destination end systems through Ethernet switch. In each destination end system, the Rxer sub module is proposed to listen to specific UDP port. Once those messages arrive, the Rxer sub module will receive and gain the contents of them. After that, the Rxer sub module will send those contents to the free Dispatcher thread and release the storage places for the following messages. After the Dispatcher sub module obtains those contents, it will then send those contents to the Assembler sub module. In the Assembler sub module, the queue is invited to storage the fragmented messages belong to one AFDX data. After the last fragmented message has been received, the original AFDX data is retrieved. This original AFDX data will then be sent to the avionics application through the Raf module. After that, the whole AFDX data transmission completed.

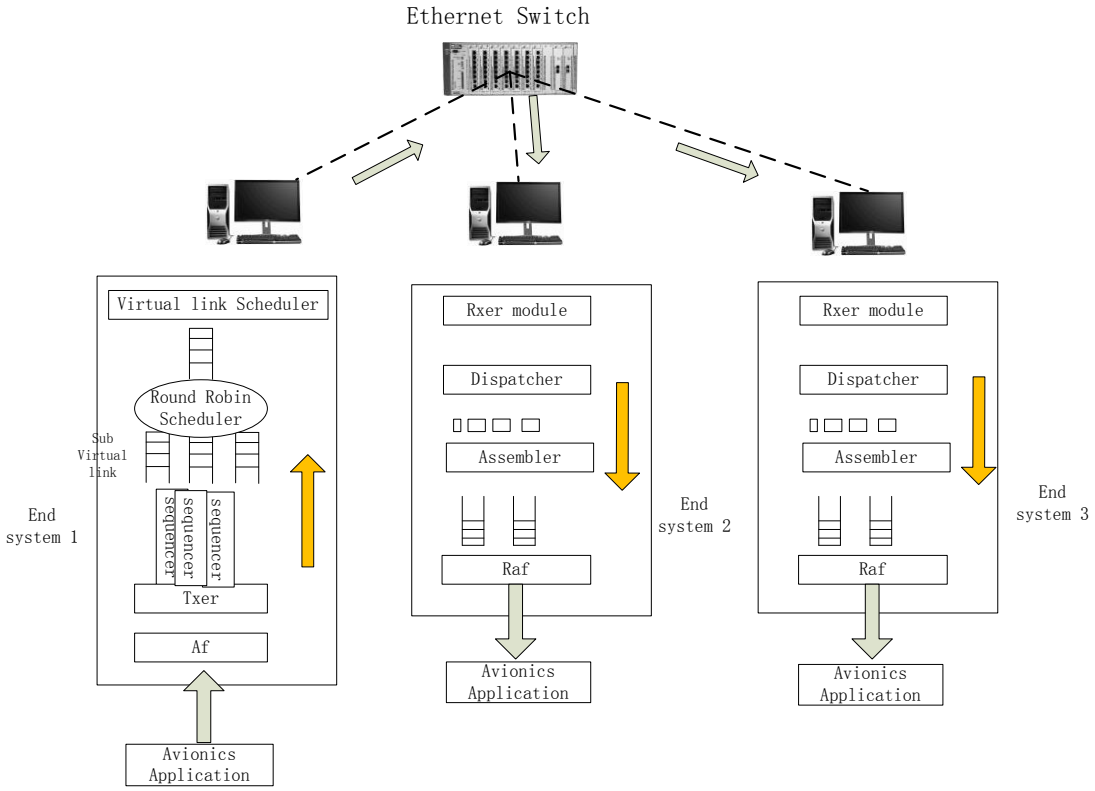


**Figure 4-2 Simulation Platform Data Exchange Behaviour**

## 4.3 Detailed Design of Avionics Application Simulation Modules

In this section, each avionics application simulation module will be elaborated.

### 4.3.1 Avionics Data Transmission Application Module

The Avionics data transmission application (ADTA) module is implemented by the author to simulate the avionics application data transmission function. This module creates the avionics data in the particular format which can be transmitted to the FACADE platform. The flow chart of this module is represented in Figure 4-3.

In this module, the socket for data transmission is created at first. Then the parameters of server are allocated. After ‘Data’ dedicated space which stored the avionics data is allocated by function intializeIM, the function addDataIM is called to move the avionics data into ‘Data’. After that, the rawdata ‘Rd’ is introduced to store ‘Data’. This rawdata can be transmitted on the AFDX network. The function innerSerializeIM is called to serialize the data ‘Data’ into the rawdata ‘Rd’. Then ‘Rd’ is sent to the FACADE platform. Once this action

completed, the ADTA module will record the transmission time and then release the allocation memory of ‘Data’ as well as ‘Rd’. As long as no ‘SIGINT’ signal received, the ADTA module would transmit the avionics data to the AFDX network continuously.

Data validation function is developed to inspect if there is any invalid AFDX data during the data transmission period. At first, the author considers applying the CRC check to this module. Since the execution duration of this module and the FACADE platform are already quite high, the author abandons this method. Then the author utilises one section of the data as the data validation part. Before each AFDX data is transferred, one serial number which sets as data check bit is copied into this AFDX data. Once this AFDX data is received by Avionics Data Reception Application which will be introduced next, this data check bit will be verified. If this check bit is not as same as it supports to be, the received AFDX data is invalid.

For the loss data detection purpose, the loss data detection function is also invited to both avionics application simulation modules and FACADE platform. At first, the author plans to use the three times handshake mechanism. Once again, due to the platform efficiency, this mechanism is not invited in this project. The author applies a simpler way to fix this issue which is the data counting. During the communication, both transmission and reception module will count the number of the data. Once the transfer completed, these numbers are displayed on the screen which can be utilised to judge if there are any missing AFDX data during the transmission section.

The functions in ADTA module are listed in Table 4-1.

**Table 4-1 Functions in ADTA Module**

| Number | API | Description |
|---|---|---|
| 1 | initializeIM | Initializing the ImplicitMessage, allocating memory for its buffer and setting its parameter to the initial values. This function does not allocate memory for the ImplicitMessage itself. It must have been |

| | | |
|---|---|---|
| | | allocated beforehand by the user |
| 2 | initializeQueue | Initializing the whole Queue, allocating memory for its buffer and setting its parameters to the initial values. This function does not allocate memory for the structure Queue itself. It must have been assigned beforehand by the user. |
| 3 | addQueue | This function adds the Queueable data to the Queue. |
| 4 | addDataIM | This function adds the data, which are pointed by data, to the ImplicitMessage. |
| 5 | destroyIM | This function frees all the previous allocated memory of the inside ImplicitMessage, but not the memory of the ImplicitMessage itself. It has to be released manually if needed. |
| 6 | destroyQueue | This function releases all the previous allocated memory of the inside Queue, but not the memory of the structure Queue itself. It has to be released manually if needed. |
| 7 | innerSerializeIM | This function converts the whole ImplicitMessage into a RawData. |
| 8 | innerSerializeQueue | This function converts the whole Queue into a RawData. |
| 9 | bareSerializeQueue | This function converts the whole Queue into a byte buffer pointed by data and returns the size by the parameter length. |
| 10 | viewQueue | This function looks for the index element inside the Queue and retrieves it, without removing it off. |

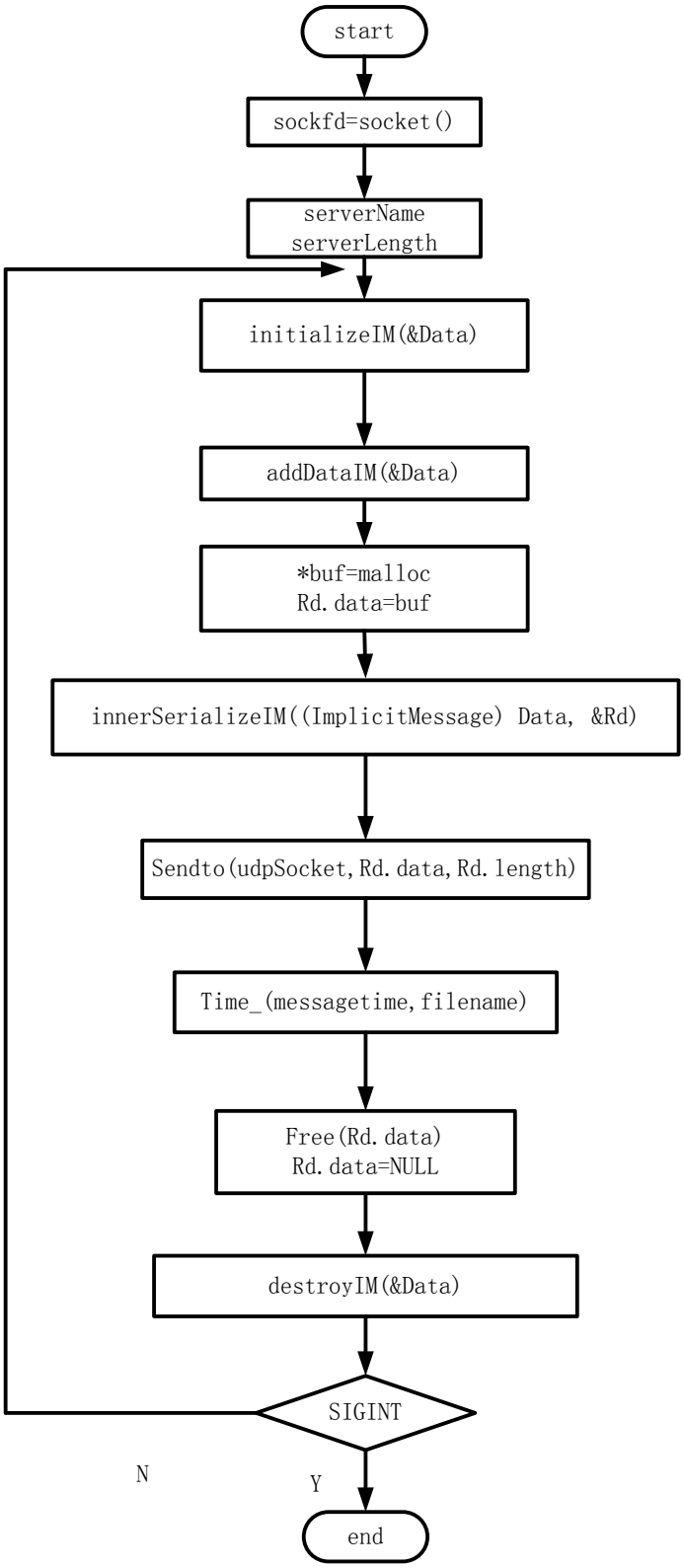


**Figure 4-3 Flow Chart of Avionics Data Transmission Application Module**

### 4.3.2 Avionics Data Reception Application Module

Avionics data reception application (ADRA) module is implemented by the author to simulate the avionics application data send function. This module receives avionics data in a particular format which is transmitted from the FACADE platform. The flow chart of this module is represented in Figure 4-4.

In this module, a socket for data reception is created and bound at first. Then the parameters of the server are allocated. After ‘Data’ dedicated space which stored the avionics data is allocated by function intializeIM, the API recvfrom is called to receive avionics data from FACADE platform to ‘buffer.' Then data in ‘buffer’ is moved into the rawdata Rd for deserialization. After Deserializing, the rawdata ‘Rd’ is copied into ‘Data’ and an array is created to obtain each data in the ‘Data’. After gaining all the data, the storage memories for both ‘buffer’ and ‘Data’ will be released. As long as no ‘SIGINT’ signal received, the ADRA module would receive the avionics data from the FACADE platform continuously.

The ADRA also has the data check function, loss data detection function and time recording and outputting function which have been elaborated before.

The functions in ADTA module are listed in Table 4-2.

**Table 4-2 Functions in ADRA Module**

| Number | API | Description |
|---|---|---|
| 1 | initializeIM | Initializing the whole Queue, allocating memory for its buffer and setting its parameter to the initial values. This function doe not allocate memory for the ImplicitMessage itself, it must have been allocated beforehand by the user |
| 2 | initializeQueue | Initializing the whole Queue, allocating memory for its buffer and setting its parameters to the initial values. But, this function does not allocate memory for the structure Queue itself, it must have been allocated beforehand by the user. |

| | | |
|---|---|---|
| 3 | addQueue | This function adds the Queueable data to the Queue. |
| 4 | addDataIM | This function adds the data, which are pointed by data, to the ImplicitMessage. |
| 5 | destroyIM | This function frees all the before allocated memory of the inside ImplicitMessage, but not the memory of the ImplicitMessage itself. It has to be released manually if needed. |
| 6 | destroyQueue | This function frees all the before allocated memory of the inside Queue, but not the memory of the structure Queue itself. It has to be released manually if needed. |
| 7 | innerDeserializeQueue | This function converts a RawData into a Queue. |
| 8 | innerDeserializeIM | This function converts a RawData into an ImplicitMessage. |

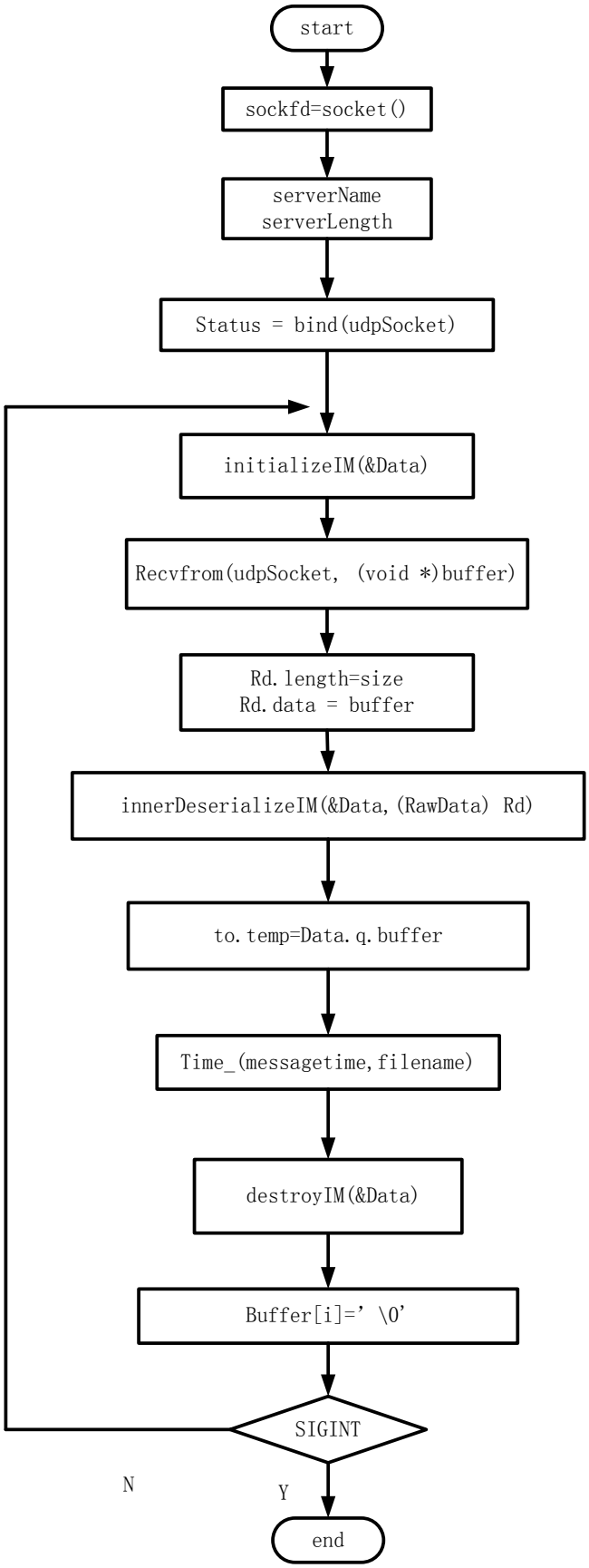


**Figure 4-4 Flow Chart of Avionics Data Reception Application Module**

## 4.4 Detailed Function Design of FACADE Platform

### 4.4.1 Af Module

In previous work, the Af module is utilised to create avionics data and to send them to Txer sub module (See Figure 4-5). Since both Af module and Txer sub module are portions of the FACADE platform, independent modules which use AFDX network for data exchanging should be introduced as avionics application. The avionics application simulations have been presented before as ADTA, ADRA. Thus, a modified Af module has been achieved by the author for receiving data from the avionics application and sending these data to subsequent AFDX module.

In this modified Af module, the UDP socket is created and bound after initialization of running environment of the Af module. Then the implicit message ‘m’ is initialized for storing avionics data from ADTA module. After the data from ADTA is received and preserved in ‘buffer’, these data are copied into the rawdata ‘Rd’ and then deserialized into the message ‘m’ for further transmission in the AFDX network. Later, the message ‘m’ is send to the Txer sub module and the transmit time is recorded. As soon as the transfer complete, the avionics data storage memory ‘buffer’ as well as message ‘m’ would be released for the next round. As long as no ‘SIGINT’ signal received, the Af module would receive the avionics data from avionics application, reformat and send these data to the AFDX network continuously.

In this project, the total time delay is defined as the transmission duration between Af module receives AFDX data from ADTA module and Raf module sends data to ADTA module.

To record the data transmission time, the time record and output function is invited. This function records the current time of transmission and output this time into a file which named after the transmission port.

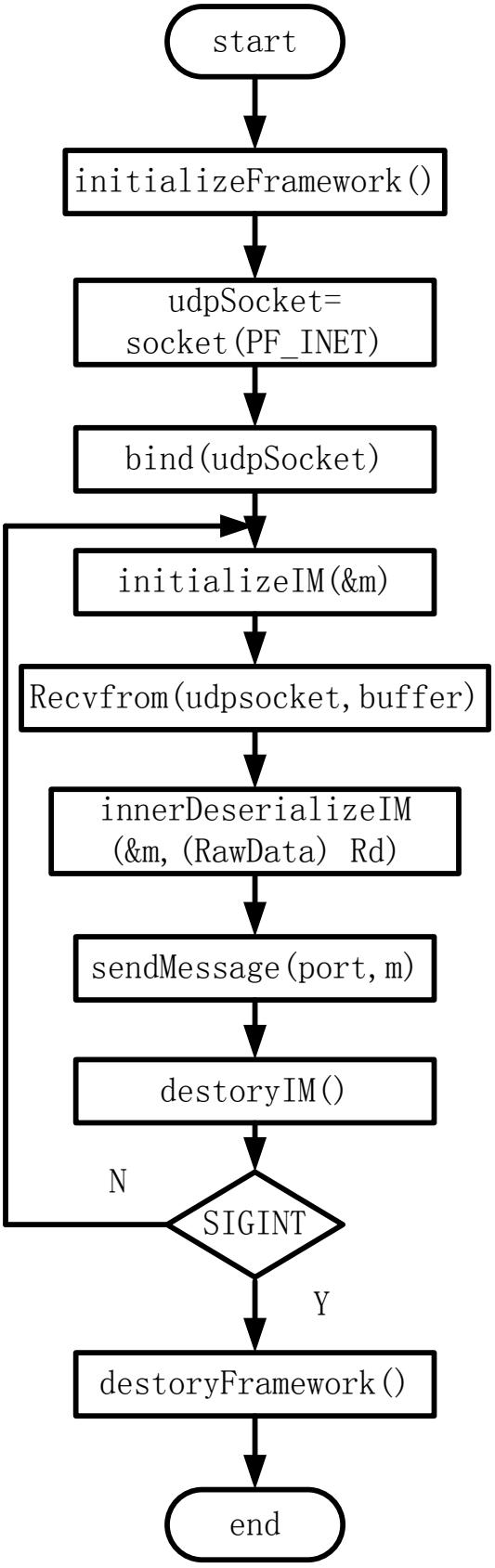


**Figure 4-5 Flow Chart of Af Module**

### 4.4.2 Raf Module

In the previous work, the Raf module is utilised to receive the avionics data and decapsulate them into a readable data. As the same reason mentioned before with Af module, in this project, a modified Raf module has been achieved by the author for receiving the AFDX data from corresponding AFDX sub module and sending these data to ADRA module (See Figure 4-6).

In this modified Raf module, the UDP socket is created after initialization of the running environment of Raf module. Then the AFDX data from the Assembler sub module are obtained. After the AFDX data have been captured and stored in the rawdata ‘Rd’, they are ready to be sent to the ADRA module. As long as no ‘SIGINT’ signal received, the Raf module would receive the AFDX data from the FACADE platform and send these data to the ADRA module continuously.

To record the data reception time, the time record and output function is invited. This function records the current time when data is sent to ADRA module and output this time into a file which named after the reception port.

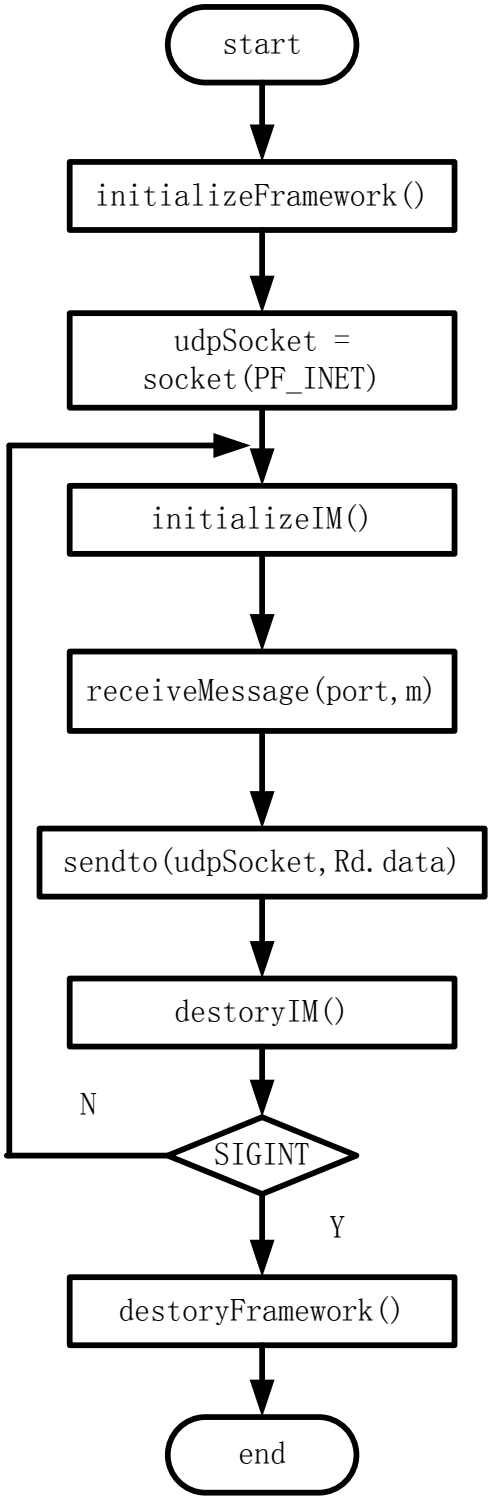


**Figure 4-6 Flow Chart of Raf Module**

### 4.4.3 CreateDB Module

The CreateDB module is proposed to create the database files which will be loaded by the corresponding modules and sub modules. Those database files should be prepared before the FACADE platform execution. This module is achieved by three main files which are main5.c, dbm.c, dbm.h. Both dbm.c and dbm.h are utilised to achieve the database operation APIs which are listed in Table 4-3. Main5.c stores the parameters of the source end systems, destination end systems as well as virtual links and uses the function listed in Table 4-3 to create the database files which contain the parameters of end systems and virtual links. After the CreateDB module execution, each end system will possess its specific database files.

**Table 4-3 Functions in CreateDB Module**

| Number | API | Description |
|---|---|---|
| 1 | writeSourceDB | Reading data of the source end systems and duplicate these data into the array of source end systems. |
| 2 | readSourceDB | Reading and obtaining data from the array of source end systems. |
| 3 | writeDestDB | Reading data of destination end systems and duplicating these data into the array of destination end systems. |
| 4 | readDestDB | Reading and obtaining data from the array of destination end systems |
| 5 | writeVLDB | Reading data of virtual links and duplicating these data into the array of virtual links |
| 6 | readVLDB | Reading and obtaining data from the array of virtual links |
| 7 | createTxerSourceDB | Reading and obtaining data from array of the source end system and create corresponding source end system database file |

| | | |
|---|---|---|
| 8 | createTxerVLDB | Reading and obtaining data from the array of virtual link and create corresponding virtual link database file |
| 9 | createTxerDestDB | Reading and obtaining data from the array of destination end system and create the corresponding destination of source end system database file |
| 10 | createRxerDestDB | Reading and obtaining data from the array of destination end system and create corresponding destination of destination end system database file |

### 4.4.4 FACADE Module

As presented before, Af and Raf modules are the interfaces of the FACADE platform while the CreateDB module generates the database files of this platform. The main AFDX network data exchange behaviour is achieved by the FACADE module. The FACADE module consists of seven sub modules which are listed below:

- Txer sub module
- Sequencer sub module
- Round Robin Scheduler sub module
- Virtual link Scheduler sub module
- Rxer sub module
- Dispatcher sub module
- Assembler sub module

All these sub modules will be represented in the following section.

#### 4.4.4.1 Txer Sub module

Txer sub module is proposed to initialize the Txer sub module, Sequencer sub module, Round Robin Scheduler sub module and Virtual Link Scheduler sub module as well as their corresponding parameters. Moreover, the Txer sub module receives the data flows from the Af module and transmit those data

flows to the idle sequencer thread. The flow chart of Txer sub module is represented in Figure 4-7.

Firstly, the Txer sub module calls the main function initializeTxer to initialize the running environment of the Txer sub module. The processes which function initializeTxer achieved are represented below:

1. Calling the API sigprocmask to block the signal ‘SIGTERM’ and ‘SIGINT’ [62], preventing the Txer sub module from unexpected interruption during the initialization period.

2. Calling the function loadConfigurationTxer to load the source end system configuration files.

3. Calling the function loadDBTxer to allocate the storage memories for parameters of the source end systems, destination end systems and virtual links. After that, the loadDBTxer will obtain these parameters by reading the database files of end systems and virtual links which are created by the CreateDB module. At the end of function loadDBTxer, the API atexit [63] is invited to release these allocated storage memories after the simulation platform exit.

4. Calling the function setEnvironmentRxer to allocate the storage memories and initialize the parameters of Rxer sub module, Dispatcher sub module and Assembler sub module.

5. Calling the function LaunchRxerTxer. This function calls the function runTxer to initiate the Rxer sub module, Dispatcher sub module and Assembler sub module. At the end of the function LaunchRxerTxer, the API atexit is invited to terminate the Rxer sub module after the simulation platform exit by using the function waitForRxer.

6. Calling the function launchRRSandVLSTxer to initiate the Round Robin Scheduler sub module and Virtual Link Scheduler sub module. The API atexit is also invited at the end of this function to terminate the Round Robin Scheduler sub module and Virtual Link Scheduler sub module after the

simulation platform exit by using the function waitForRRSandVLS.

7. Calling the function setSequencersEnv to allocate the storage memories for corresponding parameters of the Sequencer sub module which are the sequencer thread id ‘sequencer_set’ , the sequencer semaphore ‘sequencer_sem’ , the storage for messages from Txer sub module ‘petition_tray’ , and the storage for fragmented messages send to sub-virtual link ‘fmessage’ , the storage for messages of Sequencer sub module ‘in_tray’ and its mutex ‘in_tray_mutex’ , the indication for idle sequencer thread ‘free_sequencer’ and its mutex ‘free_sequencer_mutex’ . At the end of this function, the API atexit is invited to release the storage memories of the parameters of the Sequencer sub module after the simulation platform exit by using the function freeSequencerEnv.

8. Calling the function launchSequencers to initialize the Sequencer sub module. When the simulation platform exited, the API atexit is proposed to terminate the Sequencer sub module by calling the function waitForSequences.

9. Calling the function openMQTxer to create the message queue ‘txer_mq’ and its semaphore. This message queue is utilised to receive the data flows which are sent by the Af module. When the simulation platform exited, the API atexit is proposed to release this message queue.

10. Procedures 2-9 are the initializations of the Txer sub module and corresponding sub modules. After these procedures complete, the function initializeTxer reactivates the signal ‘SIGTERM’ and ‘SIGINT’ which means that the Txer sub module could be terminated since then.

Then, after the function initializeTxer complete, the function runTxer would enter the while loop. In this loop, the runTxer receives the data flows from the message queue ‘txer_mq’ and stores these data flows into the buffer ‘txer_petition’. Then the data in ‘txer_petition’ are copied into the ‘in_tray’. After then, the runTxer will search for the idle sequencer thread. Once an idle thread is found; the data in ‘in_tray’ will be sent to this thread for further operation.

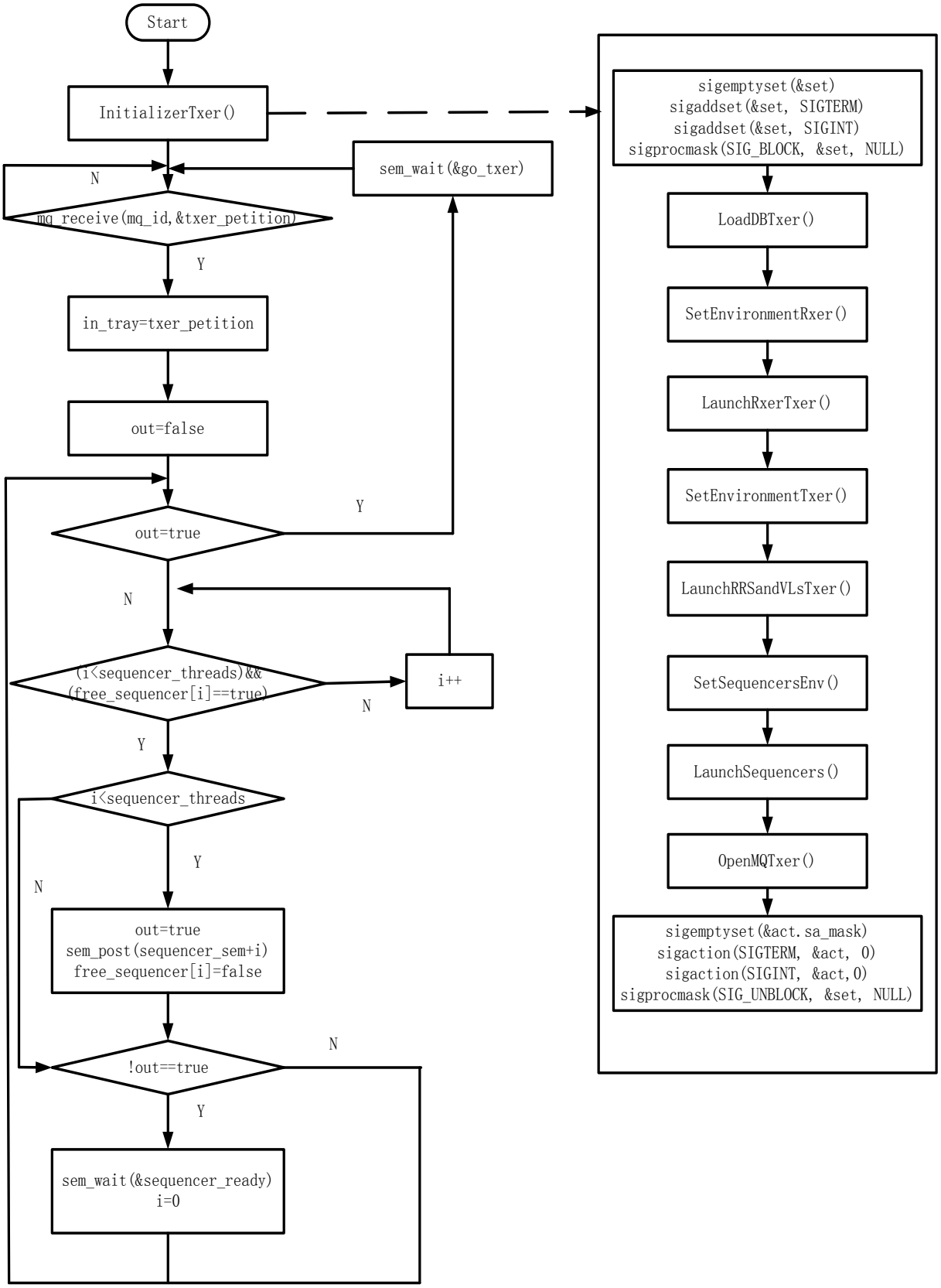


**Figure 4-7 Flow Chart of Txer Sub module**

#### 4.4.4.2 Sequencer Sub module

As mentioned before, the Txer sub module will send the processed data to the Sequencer sub module. After data arrived, the Sequencer sub module will segment the data into fragmented messages, add a sequence number to each fragmented message and send these messages to corresponding sub-virtual links. The flow chart of the Sequencer sub module is illustrated in Figure 4-8.

Firstly, the main function of the Sequencer sub module sequencer initialises ‘out_seq’ to false. This argument is utilised to indicate whether the fragmented message is the last message of the whole data which received from the Txer sub module.

Secondly, the Sequencer sub module enters the while loops and detects the value of ‘out_seq’. If the ‘out_seq’ equals to true, it indicates that the Sequencer sub module has already received the final data from the Txer sub module, the while loop will end after this execution. If the ‘out_seq’ equals to false, the Sequencer sub module will receive the data from Txer sub module buffer ‘in_tray’ and send them to the sequencer buffer ‘petition_tray’. Then the data in ‘petition_tray’ will be divided into fragmented messages following the predefined parameter $L_{max}$. The $L_{max}$ defines the largest size of the data packet which can be transmitted on the virtual link related to the parameter ‘petition_tray.port’. After that, those fragmented messages will be copied into the ‘fmessage’ and added a sequence number which ranges from 0 to 255 sequentially. Among all these sequence numbers, 0 is defined as a specific number which can only be inserted into the last fragmented message of data. This design makes it possible for the FACADE platform to recognize whether the platform has received the last fragmented message of one data. After adding the sequence numbers, the ‘fmessage’ will be sent to corresponding sub-virtual links ‘env->sub_queue’ according to the value ‘petition_tray.port’ by calling the functions subVirtualLink and addQueue. At last, the Sequencer sub module calculates the size of remaining data in the ‘petition_tray’ and assigns this value to the ‘remain_byte’s. The ‘remain_bytes’ greater than 0 means there are remaining data waiting to be fragmented. Those remaining data will cover the original data

in the ‘petition_tray’ and execute the while circulation until the ‘remain_bytes’ equals to 0.

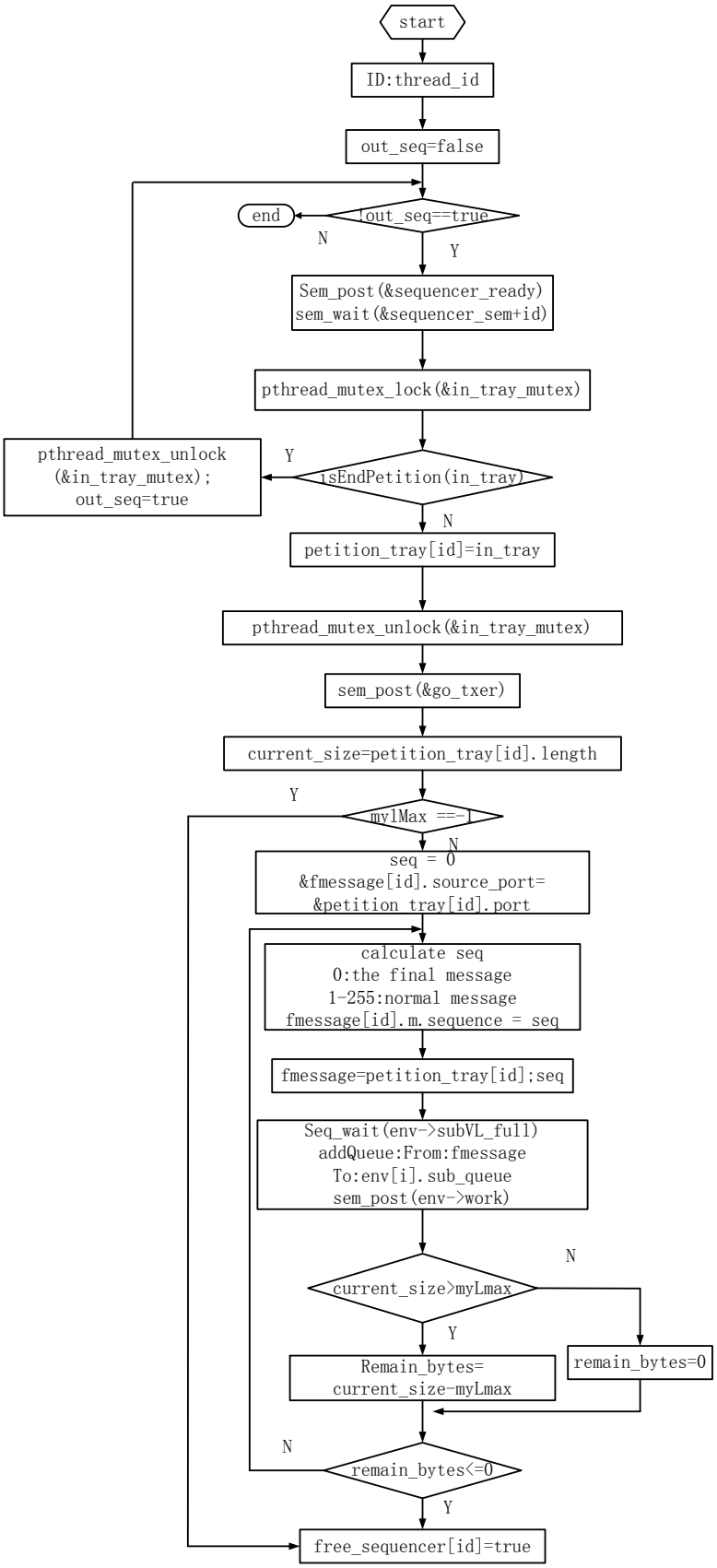


**Figure 4-8 Flow Chart of Sequencer Sub module**

#### 4.4.4.3 Round Robin Scheduler Sub module

The Round Robin Scheduler sub module bases on the round robin algorithm. It sequences the fragmented messages in their correct orders and sends them from the sub-virtual links to corresponding virtual link. The flow chart of Round Robin Scheduler sub module is represented in Figure 4-9.

Firstly, the main function of Round Robin Scheduler sub module roundRobinScheduler initialises the struct ‘env’ to store the virtual link related environment arguments. Secondly, the module will enter the do loop and the semaphore ‘env->work’ reduces one to indicate the reduction of the idle sub-virtual links ‘subvl_queues’. The ‘subvl_queues’ is introduced to store the data waiting to be sent to corresponding virtual link. Thirdly, the module estimates whether the ‘subvl_queues’ is empty. If ‘subvl_queues’ is empty, the data in this sub-virtual link have been transmitted completely which means this sub-virtual link is idle. The argument count will add one to indicate the rise amount of the unused sub-virtual links. If the ‘subvl_queues’ is not empty, the data in ‘subvl_queues’ will be transmitted into the ‘subvl_m’. Then the module will send the data in ‘subvl_m’ to corresponding virtual link ‘env->vl_queue’.

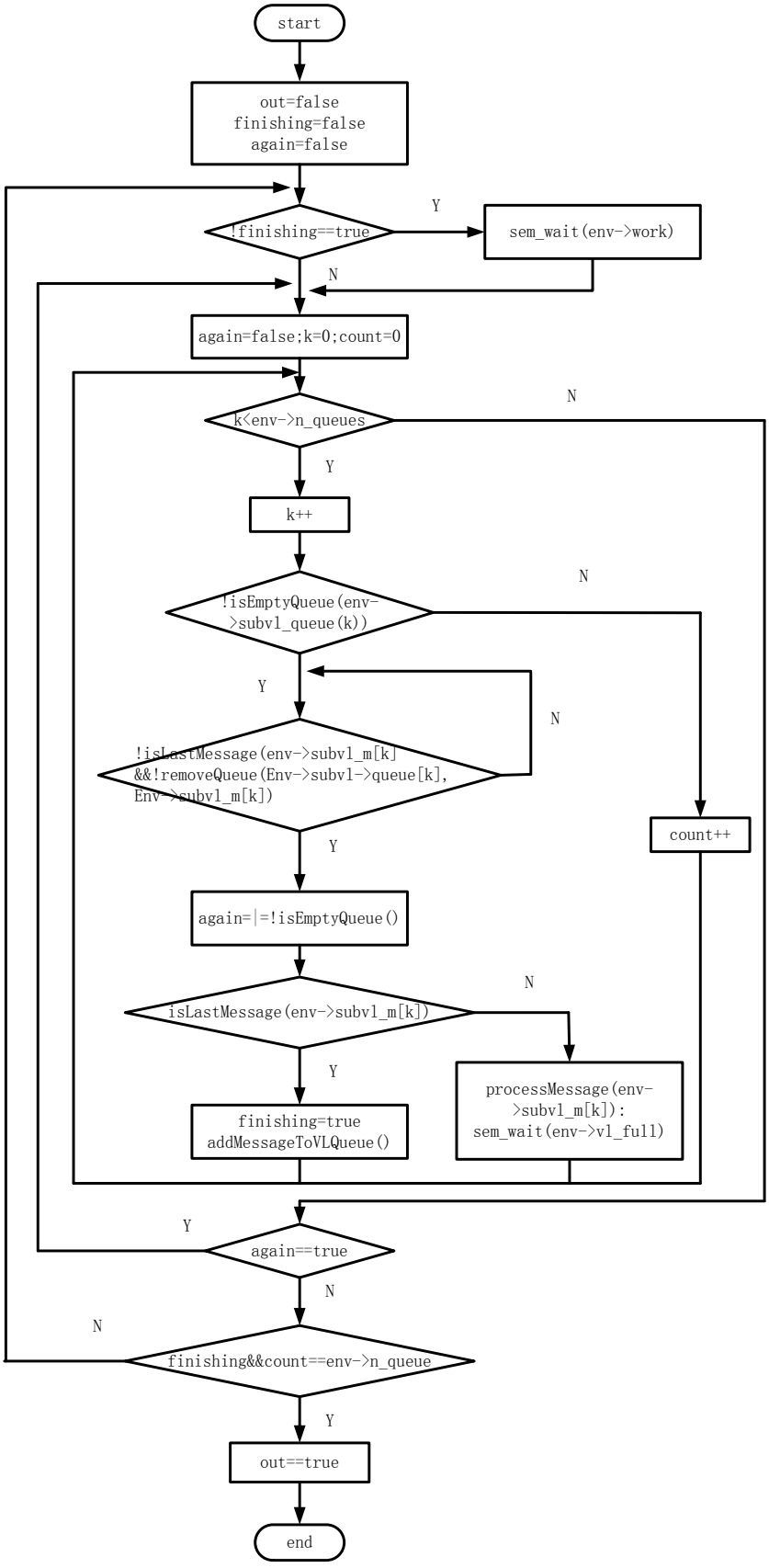


**Figure 4-9 Flow Chart of Round Robin Scheduler Sub module**

#### 4.4.4.4 Virtual Link Scheduler Sub module

The Virtual Link Scheduler sub module transmits the data packets following the time interval BAG which is predefined in the virtual link database files. This sub module sends the data packets to the physical link. These data packets will be transmitted to the corresponding UDP port of destination end system by network devices. The flow chart of Virtual Link Scheduler sub module is illustrated in Figure 4-10.

Firstly, the Virtual Link Scheduler sub module initializes the struct ‘env’ which is proposed to store the virtual link related environment arguments. Secondly, the module read the BAG value ‘vls.bandwidthAllocationGap’ of the chosen virtual link which will transmit the data packets later. The sub module will send the data packets following the BAG as its time interval. Lately, the Virtual Link Scheduler sub module creates a UDP socket which will then be utilised to send the data packets to the destination end system. After all operations mentioned above has been completed, the sub module will enter the do loop and exit the loop till the parameter ‘out’ equals to true.

In this ‘do’ loop, first data of the virtual link queue ‘vl_queue’ will be copied into the fragmented message ‘vl_m’. The ‘vl_m’ is introduced to store the fragmented messages which are processed by the Virtual Link Scheduler sub module. If the message in ‘vl_m’ is not the final message of all messages, the sub module will copy the data from the ‘vl_queue’ to the ‘vl_m’ successively. Then the data in ‘vl_m’ will be sent to corresponding destination end system by UDP socket. If the message in ‘vl_m’ is the final message, the message has already been forwarded to the last loop. Then the parameter out becomes true, the ‘do’ loop terminates.

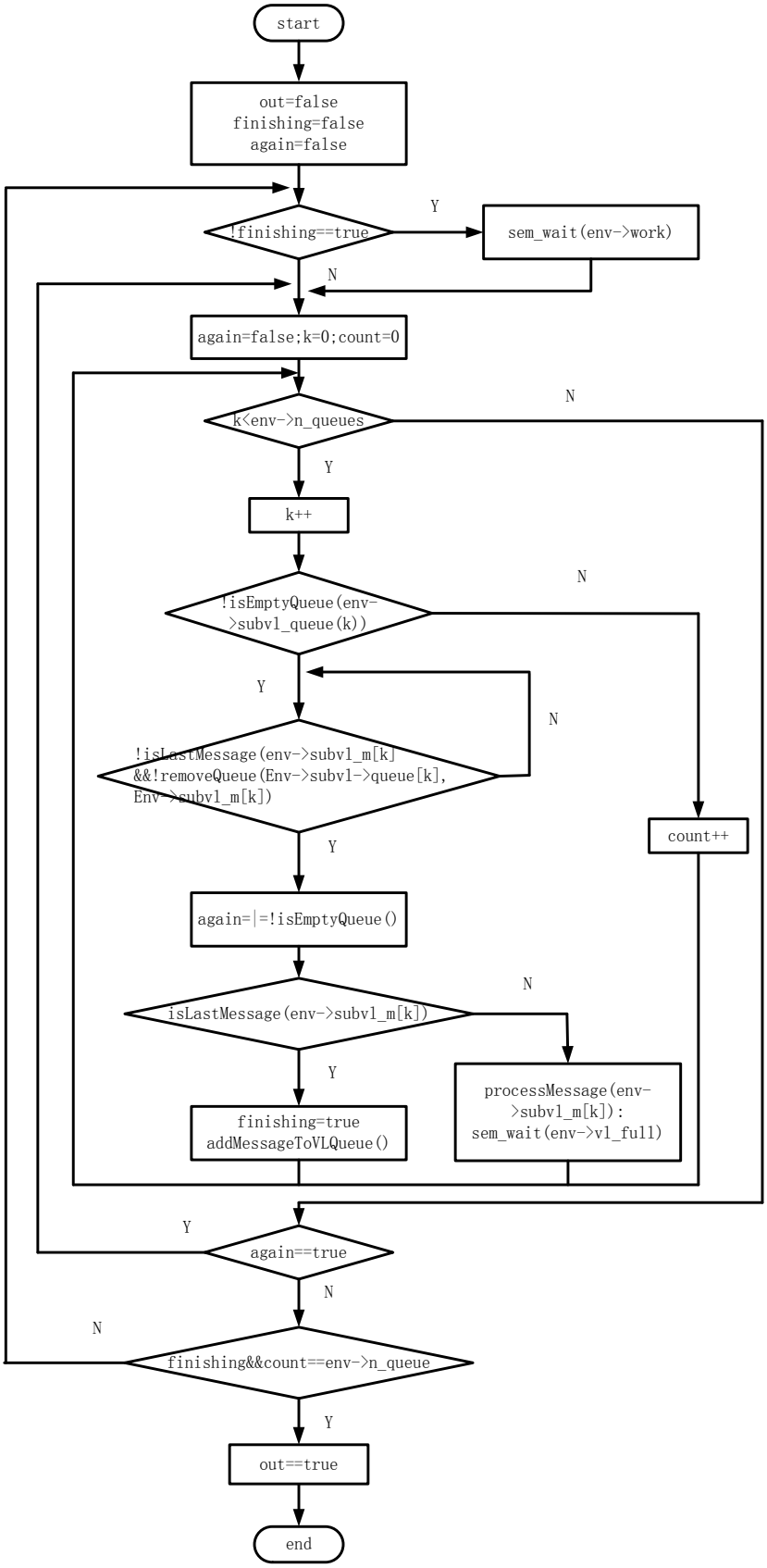


**Figure 4-10 Flow Chart of Virtual Link Scheduler Sub module**

#### 4.4.4.5 Rxer Sub module

As mentioned before, the Rxer sub module is invoked by the Txer sub module using the function LaunchRxerTxer. This sub module is proposed for the destination end system to receive the data flows transferred from the source end system. The flow chart of the Rxer sub module is shown in Figure 4-11.

The operations achieved by the Rxer sub module are represented as below:

1. The API sigprocmask is called to unblock signal ‘SIGTERM’ and ‘SIGINT’ which have been blocked by the Txer sub module when initializing the Rxer sub module. Then the Rxer sub module can be terminated since then.

2. Calling the function loadDBRxer to load the configuration file of the destination end system. Firstly, the loadDBRxer will allocate the storage memories to the key and port of destination end systems. Secondly, the loadDBRxer will obtain these two parameters from the configuration file and store them in the allocated storage memories.

3. Calling the function setEnvironmentRxer which has already been introduced in the Txer sub module section.

4. Calling the function setDispatcherEnv to allocate the storage memories and initialise the assembler queue ‘assembler_queue’, the assembler thread Mutex ‘assembler_queue_mutex’, the assembler thread resource semaphore ‘assembler_full’ and the assembler thread running state semaphore ‘assembler_work’.

5. Calling the function launchDispatcher to boot the Dispatcher sub module.

6. Calling the function openAssemblerMQ to allocate the storage memories and initialize the message queue ‘AFDXport_id’ and the message queue semaphore ‘AFDXport_sem’.

7. Calling the function launchAssembler to boot the Assembler sub module.

8. Calling the function server. Firstly, the function server will create and bind the UDP socket ‘in_port_set’. This UDP socket is utilised to receive the data

flows from the source end system. Then the received data are stored in the ‘rxer_message’. Later, the server function will obtain the data sequence number ‘sequence’, data of the data flows ‘data’, the destination port of destination end system ‘dest_port’, the source port of the data flow ‘source_port’, the source IP address of the data flow ‘ip_address’ and then copy them into the ‘rxer_message’. At last, function server will search the idle Dispatcher thread and send the received data to it.

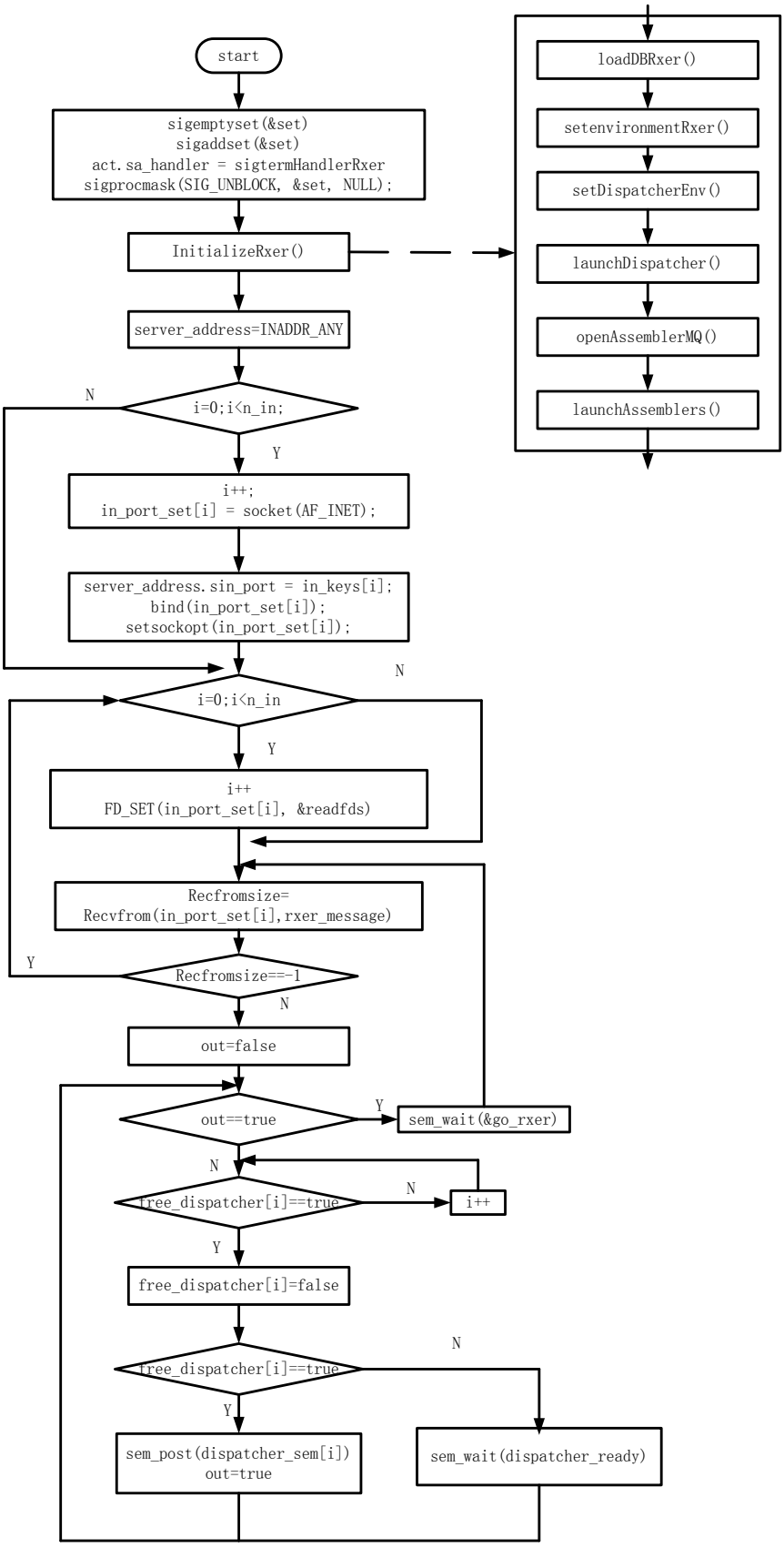


**Figure 4-11 Flow Chart of Rxer Sub module**

#### 4.4.4.6 Dispatcher Sub module

The Dispatcher sub module is introduced to receive the fragmented message from the Rxer sub module. After obtaining specific values from this message, those values will be sent to the Assembler sub module.

Firstly, the Dispatcher sub module main function dispatcher enter the while loop. Then Dispatcher sub module will estimate whether the fragmented message in ‘rxer_message’ is the final message. If it is the final message, the argument out_dis will turn to true and the loop will terminate after this execution. If the message in ‘rxer_message’ is not the last message, The Dispatcher sub module will obtain destination port, data, the length of data as well as the sequence number of this fragmented message and copy those data into ‘dispatcher_message’. Later, the ‘dispatcher_message’ will be transmitted to the Assembler sub module queue ‘assembler_queue’.

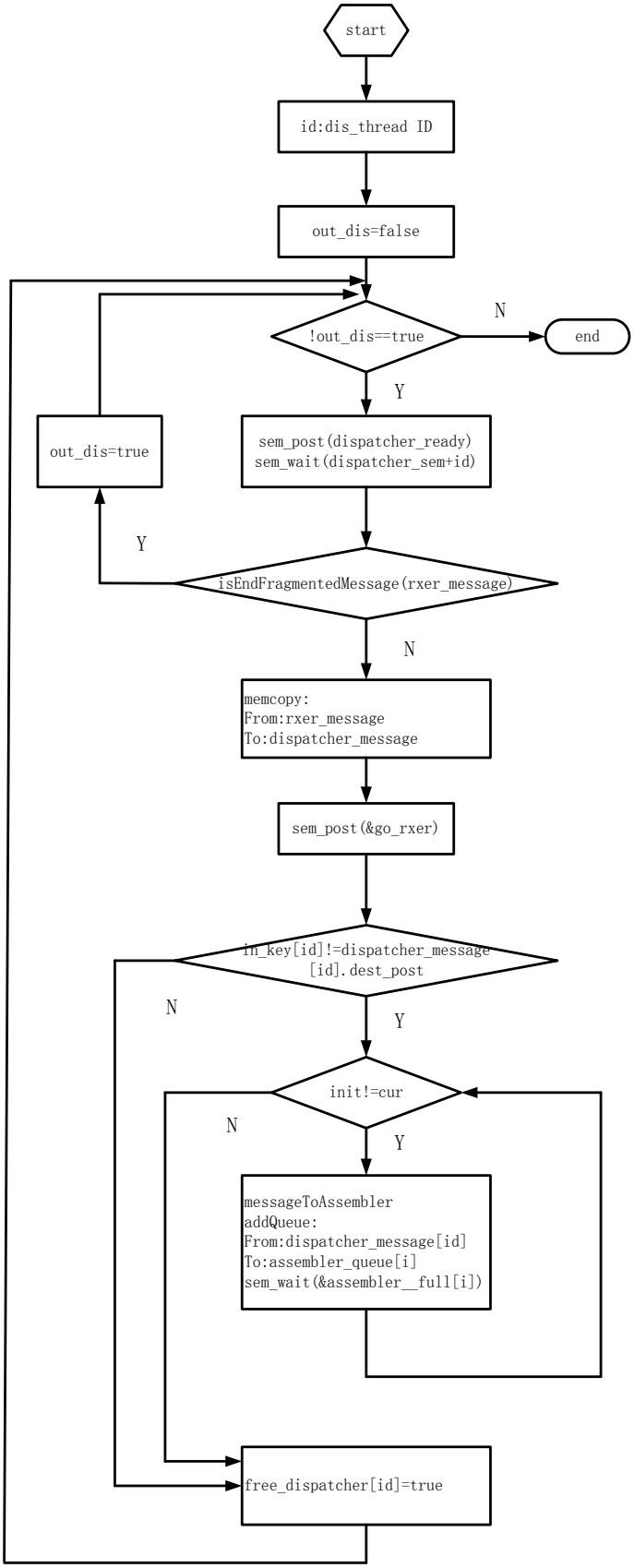


**Figure 4-12 Flow Chart of Dispatcher Sub module**

#### 4.4.4.7 Assembler Sub module

The Assembler sub module receives the fragmented messages from the Dispatcher sub module and stores these messages in the ‘buffer’. After all the fragmented messages belong to one particular message have been received completely, those fragmented messages in the ‘buffer’ will be assembled and transferred into the original message and then sent to the Raf module by using a message queue.

Firstly, Assembler sub module main function assembler enters the do loop. Then in this loop, the pending fragmented messages in the ‘assembler_queue’ will be copied into ‘fagm’. After that, the Assembler sub module will estimate whether ‘fagm’ is the final message. If the message in ‘fagm’ is the last message which means the transmission has been completed, the argument out turns to true and the do loop will terminate after this execution. If the message in ‘fagm’ is not the final message, the fragmented message in ‘assembler_queue’ will be copied into the ‘buffer’, waiting for data assembly. Then the Assembler sub module will detect the value of ‘fagm.m.sequence’ to estimate whether the ‘buffer’ has received all the fragmented messages of one data. If the transmission is completed, the fragmented messages in the ‘buffer’ will be sent as a whole message to the Raf module through the message queue. If not, the ‘buffer’ will send the stored data till the Assembler sub module receives all the fragmented messages.

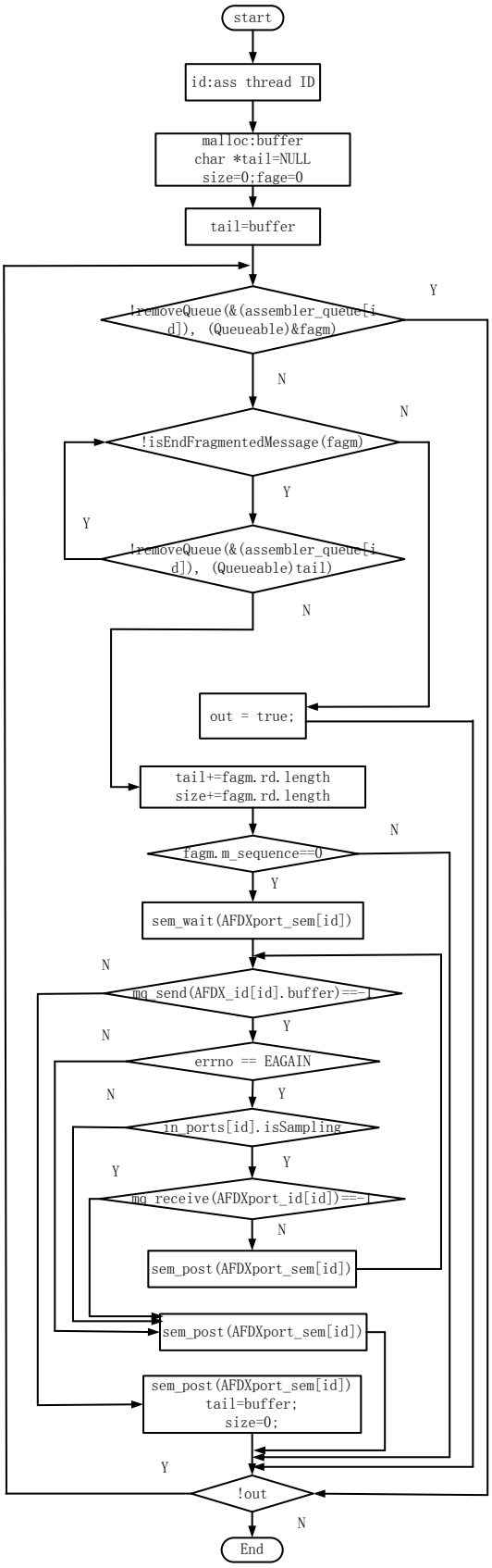


**Figure 4-13 Flow Chart of Assembler Sub module**

## 4.5 Execution of Simulation Platform

### 4.5.1 Execution of FACADE Platform

In the compilation section, the makefile is invited to compile the simulation platform which can increase the efficiency of the compilation. After the makefile has been run, several executable files are generated which are CreateDB, afdx, af, raf. The CreateDB file is utilised to execute the database module while the afdx file is proposed to run the FACADE module. Af and raf files are invited to execute the Af module and Raf module respectively.

The execution steps and functions of each command of the FACADE platform are listed below:

- ./make clean: to delete the previous executable files and database files
- ./make: to create the executable files.
- ./createDB: to create the database files.
- ./afdx b <ES number> <Database file version>: to execute the FACADE module of the simulation platform.
- ./af <Data sending port> <ES number> <DB version> <value> <Local data exchange port>: to execute the Af module.
- ./raf <Data receive port> <ES number> <DB version> <Server data exchange port>: to execute the Raf module.

About each end system, the fourth section of the IP address is defined as ES number. Since the database files are generated every time before the whole simulation platform execution, the ‘DB’ version is 1. The ‘Value’ is defined as 0 for reservation utilisation. The ‘Local data exchange’ port and ‘Server data exchange port’ are the UDP ports which are proposed to communicate with avionics applications. These two ports can be any ports except those busy ports occupied by operation systems or other processes.

Ctrl +c should be applied to terminate the FACADE platform. The FACADE module should be terminated before other modules.

### 4.5.2 Execution of Avionics Application Simulation Modules

In the compilation of avionics application simulation modules, the makefile is also invited to compile these modules. After the makefile has been run, two executable files generated which are linux-client and linux-server. The linux-client is proposed to send the AFDX data to the FACADE platform. The linux-server is invited to receive the AFDX data from the FACADE platform.

The execution steps of the avionics application simulation modules are listed below:

- ./make clean- to delete the previous avionics application simulation modules
- ./make- to create the avionics application simulation modules
- ./send <server-port>- to execute the linux-client.
- ./receive <local-port>- to execute the linux-server.

The server-port of linux-client application should be as same as Local data exchange port of the Af module. The local-port of linux-server application should be as same as server data exchange port of the Raf module.

## 4.6 Validation of Platform

In this section, avionics application simulation modules and FACADE platform will be verified. To simplify the verification, only two end systems are involved in this test which IP addresses are 192.168.1.8 and 192.168.1.3.

Figure 4-14 shows the preparation of the FACADE platform and Avionics application simulation modules of 192.168.1.8. As shown in figure, five windows are represented. The left window on the top is the Af module which is utilised to receive AFDX data from avionics application simulation module and then transfers those data to the FACADE platform. The left window on the bottom is the Raf module which is utilised to receive the data from FACADE platform and then sends those data to the avionics application simulation module. The right window on the top shows the execution the states of FACADE platform which is the FACADE module. The right window in the middle represents the ADTA module of avionics application simulation module which is utilised to send the AFDX data to the FACADE platform. The right window on the bottom shows the

ADRA module of avionics application simulation module which is utilised to receive the AFDX data to the FACADE platform.

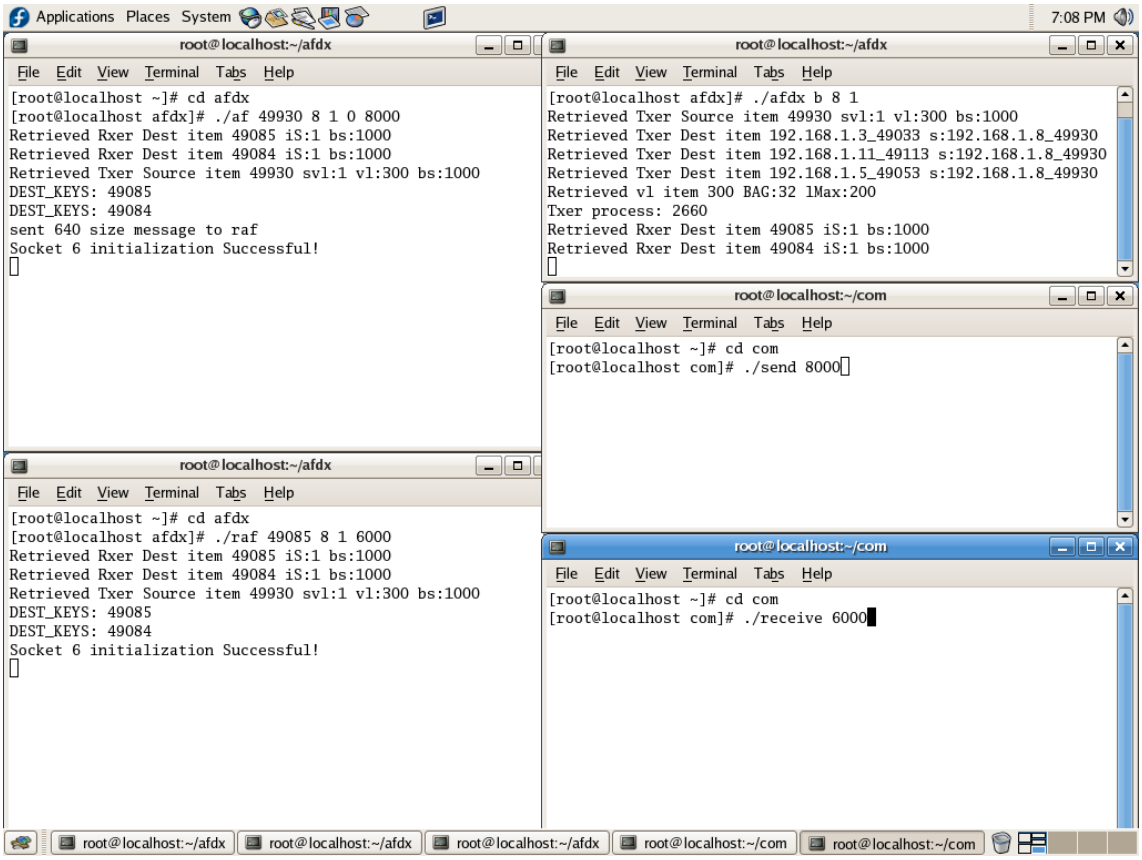


**Figure 4-14 Preparation of FACADE and Avionics Application Simulation platform**

Figure 4-15 illustrates the data change execution of the simulation platform and two modules. The functions of those windows in Figure 4-15 are as same as them in Figure 4-14 which will not be elaborated again. As can be seen from Figure 4-15, each module is working properly. After the AFDX data are sent or received by each module, all of them will display the states in the windows. The data check function and data counting function are both represented by each module.

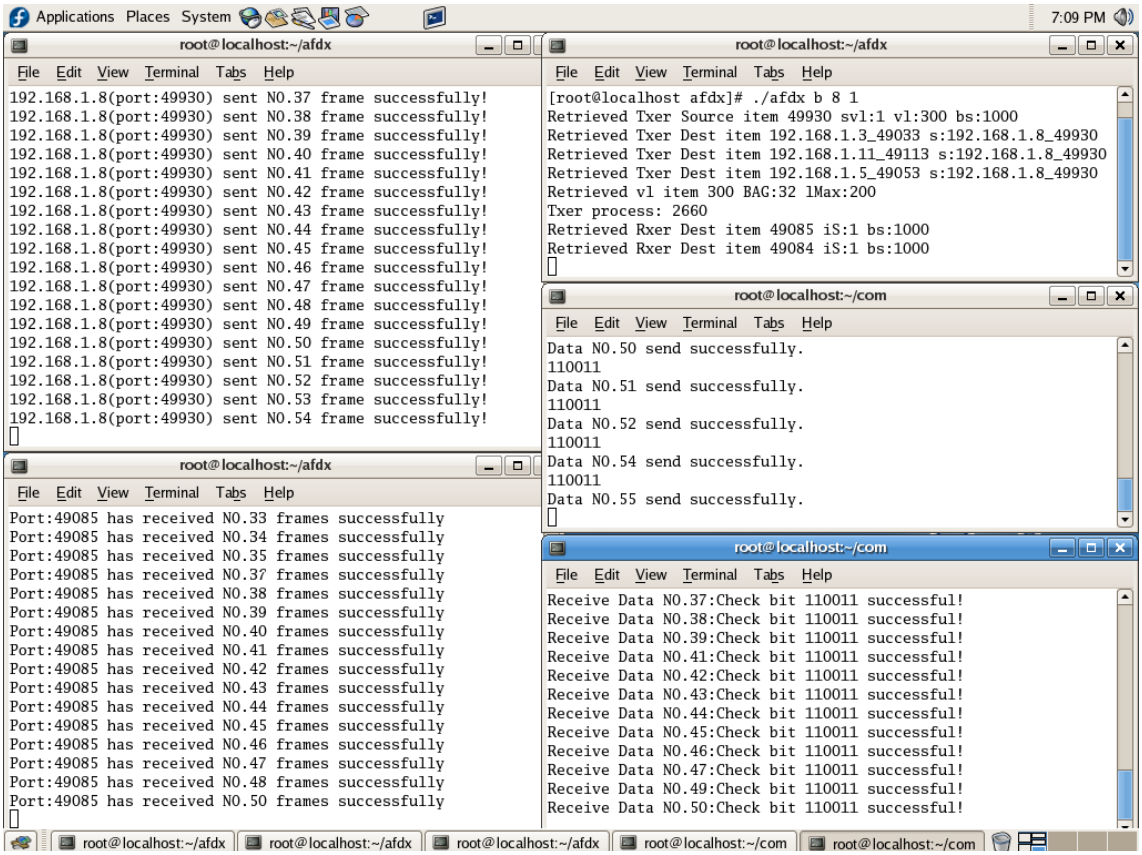


**Figure 4-15 Data Exchange of FACADE and Avionics Application Simulation Platform**

## 4.7 Summary

In this chapter, the architecture of the FACADE platform is first represented. Secondly, the data exchange behaviour of the simulation platform is then illustrated. Thirdly, the detailed design of avionics application simulation modules is introduced. In this section, both ADTA module and ADRA module are elaborated. Then the detailed design of the FACADE platform is represented. In this part, the Af module, Raf module and CreateDB module are shown at first. After that, the FACADE module is elaborated. This module achieved the main functions of the AFDX network and is the main part of the FACADE platform. In the FACADE module, seven sub modules consist of the whole platform are shown. Next, the execution steps of both AFDX network simulation platform and avionics application simulation modules are listed as

well as notice when terminating the FACADE platform. Finally, the verification of these two simulation platforms is represented.

# 5 EXPERIMENT DESIGN AND EXECUTION

## 5.1 Introduction

The FACADE platform is proposed to research the associations between the total time delay and other variances which are $L_{max}$, BAG, the amount of destination end systems in one specific virtual link, the amount of virtual links in one particular network and the amount of switches one specific virtual link traverses. According to the delay formula obtained in chapter 3, the variables $L_{max}$, BAG, the amount of virtual links in AFDX network and the amount of switches one specific virtual link traverses could affect the performance of total time delay while the amount of destination end systems in one specific virtual link will not. To valid these, five experiments are designed.

In the experiment section, five computers executing under Fedora system are utilised as the end systems. Two switches, which models are HP Procurve 4000M, are proposed as the network devices. To obtain the total time delay, the experiments are designed to detect the time interval of the AFDX data transmission between the source end system and the target destination end system. At the very beginning, the author plans to use the Linux command ‘time.' This command can obtain the execution duration of the system time and user time of one command. After the trail, the author realizes that although the period of transmission of the source end system is obtained as well as the duration of reception of the target destination end system, these data can only be utilised to calculate the mean time duration of the transmission or reception respectively. The duration of every AFDX transmission data can’t be gained from those mean values. Since the total time delay can be obtained by calculating the time difference between the transmission time and reception time, the time recording function has been invited to the simulation platform and validated. This function has been presented in chapter 4.

## 5.2 Time Synchronization

Since the time recording function has already been proved working properly, the following issue is how to synchronize the time of end systems utilised in these experiments.

The author tries two different approaches to achieve this goal. The first approach is the automatic time synchronization. The Fedora system can synchronize time regularly. The NTP service is utilised to synchronize the time. In this case, the time synchronization is only needed for these five computers while the internet time is not requisite. Then the computer 192.168.1.14 is chosen as a time server while other computers are the clients. The steps of time synchronization are listed below:

- rpm –qa | grep ntp - to detect the installation information of Operating System service NTP. If the NTP service has not been install yet, then the installation CD should be prepared to install this function.
- chkconfig ntpd on - to configure the automatic start-up of the NTP service when the operating system boots.
- ps –ef | grep ntp - ntp - to check whether the NTP service is running properly.
- service ntpd start - to start the NTP service if it is not running.

To configure the time server, the file ntp.conf under file directory /etc should be modified. Thus, the file on computer 192.168.1.14 should be modified. The modifications of file ntp.conf on computer 192.168.1.14 are represented below:

- Note the ‘restrict default ignore’- to enable the NTP server function.
- Input ‘restrict 192.168.1.0 mask 255.255.255.0’- to permit the computers in 192.168.1.0/24 to update time with the time server 192.168.1.14.

Till now, the modification of the file ntp.conf has been completed. Then the NTP service of 192.168.1.14 should be restarted by using the following commands:

- /etc/init.d/ntpd restart - to restart the NTP service.
- chkconfig ntpd on - to configure the automatic start-up of the NTP service when operating system boots.

- Add TCP port 123 and UDP port 123 in the firewall configuration.

Since the time server has been configured completely, the modifications on time client are listed below:

- Note the ‘server 127.127.1.0’ and ‘fudge 127.127.1.0 stratum 10’- to stop the computer synchronizing time with the local hardware.
- Input ‘server 192.168.1.14’ - to synchronize time with the time server 192.168.1.14.
- In the file crontab under file directory /etc, input ’01 * * * * root ntpdate –u 192.168.1.14’- to synchronize time with 192.168.1.14 every hour. Due to the operating systems version, the Fedora system applied in the experiments can’t execute this automatic task less than one hour. That means one hour is the smallest time interval of this automatic execution.

The configurations of time client have been completed so far. As long as the connection between time server and time client exists, the time synchronization should be executed every hour.

Although the time server and client have been configured, several issues have been discovered during the test. One is that the automatic synchronization of several computers does not work normally. Those computers can’t obtain the correct time from the time server which make the calculated total time delay values not accurate. The author considers that the computers applied for experiments are too old which have already been used for more than ten years. The version of operating system is Fedora 5 which has been conducted for more than a decade. These two reasons could cause the abnormal of automatic time synchronization.

The other issue is that even time automatic synchronization is working normally, the synchronization frequency is once every hour. As mentioned before, those computers have been used for more than ten years. The clock of them is not working properly. Even during the duration between the successive time synchronization, the time deviation can up to second. This time deviation is unacceptable in these experiments.

Since the issues represented above, the automatic time synchronization can't be invited in these experiments. Then the manual time synchronization is proposed. Before each experiment, the time synchronizations between clients and time server are executed manually. The command is 'ntp –u 192.168.1.14'. This command runs the time synchronization when the command is executed. The time deviation is then deduced to milliseconds. This deviation is considered acceptable for these experiments.

## 5.3 Experiment Detailed Design and Total Time Delay Calculus

In the experiment design section, five experiments are designed. Those experiments are utilised to research the associations between the total time delay and other variables which are $L_{max}$, BAG, the amount of end systems in one specific virtual link, the amount of virtual links in one specific network and the amount of switches one specific virtual link traverses. Five experiments are designed to achieve these goals. Moreover, the experiment results will be compared with calculus results to verify the FACADE platform. The approach applied in the experiments is variable control method. The detailed designs of these five experiments are elaborated next.

### 5.3.1 Experiment 1(Variable $L_{max}$)

This experiment aims to study the association between total time delay and $L_{max}$. The illustration of this experiment is represented in Figure 5-1.

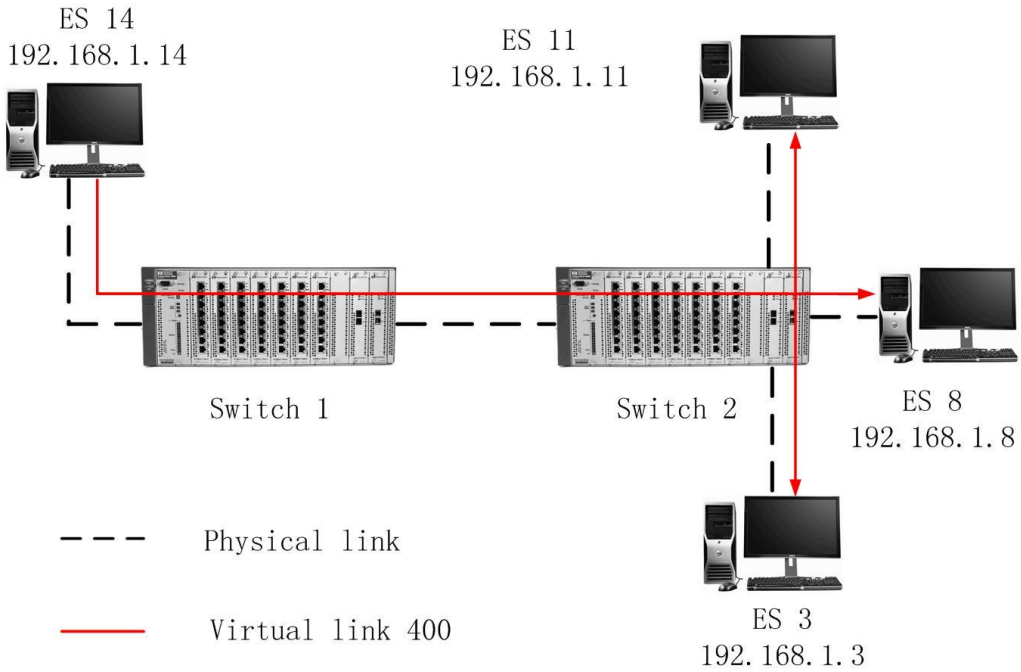


**Figure 5-1 Diagram of Experiment 1**

As shown in Figure 5-1, four computers which IP addresses are 192.168.1.14, 192.168.1.11, 192.168.1.8 and 192.168.1.3 are applied. These computers are proposed as the end systems. The mark ES is the end system id of this computer. For instance, ES 14 is the end system id of the computer which IP address is 192.168.1.14. Two switches, switch 1 and switch 2, are proposed as the AFDX network switches. The black dotted line in the figure represents the physical links between the end systems and switches. The solid red line in the figure represents the virtual link amount the end systems. In this experiment, the ES 14 is invited as the source end system while other three end systems are destination end systems. Only one virtual link exists in this experiment which virtual link id is 400, and it has a unique sub-virtual link. The BAG value of this virtual link is 32ms. In this experiment, $L_{max}$ of the virtual link 400 is set as the variable which ranges from 100 bytes to 700 bytes, raise 100 bytes each time. This experiment will record the transmission time of FACADE platform on the ES 14 and the reception time of FACADE platform on the ES 8. These two values are utilised to calculate the total time delay between these two end systems. As the $L_{max}$ changes, the experiment will be executed seven times

while seven groups of data will be gained. Those data will then be analysed to detect the association between $L_{max}$ and total time delay.

The total time delay of experiment 1 is represented in Table 5-1.

**Table 5-1 Total Time Delay of Experiment 1**

| $L_{max}$/bytes | 100 | 200 | 300 | 400 | 500 | 600 | 700 |
|---|---|---|---|---|---|---|---|
| Time/ms | 256.64 | 128.512 | 96.528 | 64.448 | 64.544 | 64.64 | 32.368 |

### 5.3.2 Experiment 2 (Variable BAG)

This experiment is designed to study the association between total time delay and BAG. The illustration of this experiment is represented in Figure 5-2.

As shown in the figure, in this experiment, ES 14 is invited as the source end system while other three end systems are destination end systems. Only one virtual link exists in this experiment which virtual link id is 400, and it has a unique sub-virtual link. The $L_{max}$ value of this virtual link is 200 bytes. In this experiment, the BAG of the virtual link 400 is set as the variable which ranges from 2ms to 64ms, increasing each time continuously. This experiment will record the transmission time of FACADE platform on the ES 14 and the reception time of FACADE platform on the ES 8. These two values are utilised to calculate the total time delay between these two end systems. As the BAG values change, the experiment will be executed six times while six groups of data will be gained. Those data will then be analysed to detect the association between BAG and total time delay.

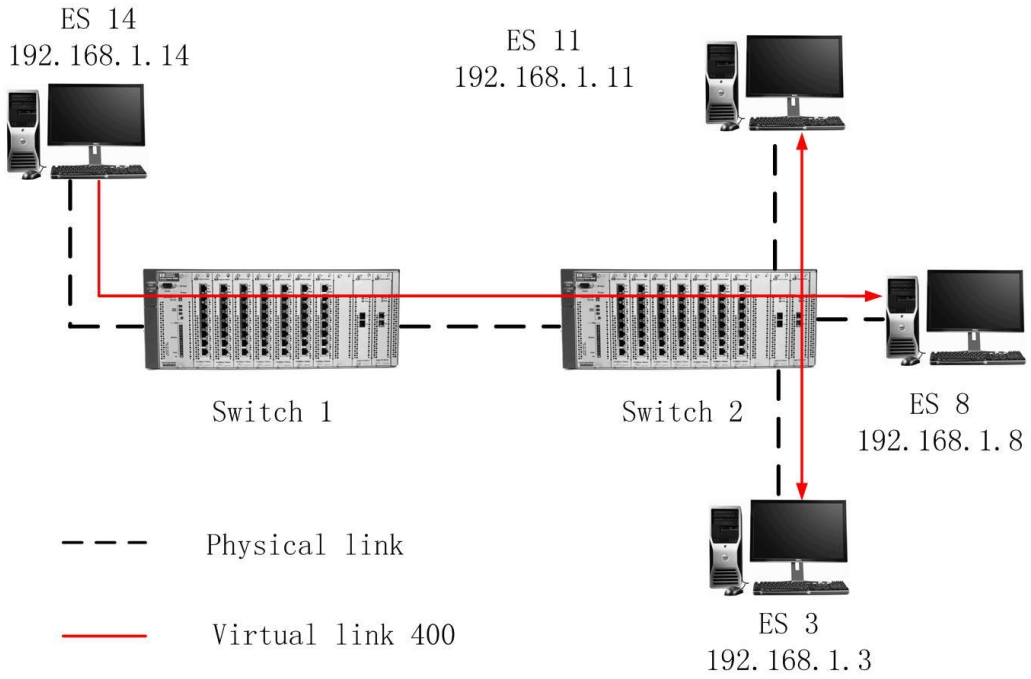


**Figure 5-2 Diagram of Experiment 2**

The total time delay of experiment 2 is represented in Table 5-2.

**Table 5-2 Total Time Delay of Experiment 2**

| BAG/ms | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|
| time/ms | 8.512 | 16.512 | 32.512 | 64.512 | 128.512 | 256.512 |

### 5.3.3 Experiment 3 (Variable Amount of Destination End Sources)

This experiment is designed to study the association between total time delay and the amount of destination end systems in one specific virtual link. The illustration of this experiment is represented in Figure 5-3.

As illustrated in the figure, in this experiment, ES 14 is invited as source end system while other four end systems are destination end systems. Three virtual links exist in this experiment which virtual link id is 400, 410 and 420. Virtual link 400 and virtual link 420 have one sub-virtual link. The virtual link 400 is represented as the solid red line while the virtual link 420 is shown as the solid purple line. The virtual link 410 illustrated as the dark blue solid line has two sub-virtual links. Those two sub-virtual links belong to virtual link 410 are

represented as solid light blue line and solid green line respectively. The $L_{max}$ values of these three virtual links are all 200 bytes and the BAG values are all 32ms. In this experiment, the amount of destination end systems in one specific virtual link is set as the variable. The virtual link 400 contains three destination end systems which are ES 11, ES 8 and ES3. The virtual link 410 has two sub-virtual links which ids are 1 and 2. The sub-virtual link 1 has three destination end systems which are ES 11, ES 8 and ES 3 while the sub-virtual link 2 has three destination end systems which are ES 11, ES 5 and ES 3. That means the virtual link 410 has six destination end systems in one virtual link. The virtual link 420 has four destination end systems which are ES 11, ES 8, ES 3 and ES 5. This experiment will record the transmission time of FACADE platform on ES 14 and the reception time of FACADE platform on ES 8. These two values are utilised to calculate the total time delay between these two end systems. As only one virtual link executing each time, the experiment will be executed three times while three groups of data will be gained. Those data will then be analysed to detect the association between the amount of destination end systems in one specific virtual link and total time delay.

The total time delay of experiment 3 is represented in Table 5-3.

**Table 5-3 Total Time Delay of Experiment 3**

| Scenario | 1vl<br>3 destination end systems | 1vl<br>4 destination end systems | 2 sub-vl in 1vl<br>6 destination end systems |
|---|---|---|---|
| Time/ms | 128.512 | 128.512 | 128.512 |

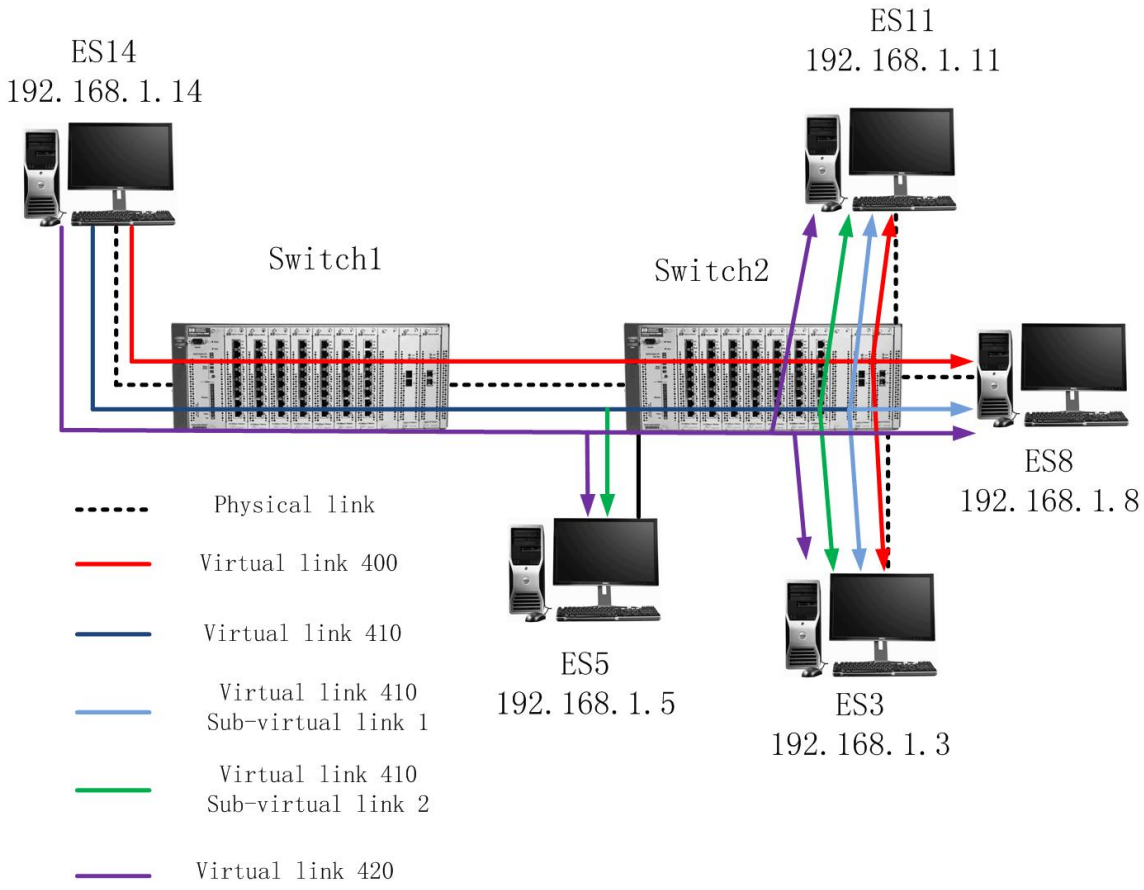


**Figure 5-3 Diagram of Experiment 3**

### 5.3.4 Experiment 4 (Variable Amount of Virtual Links)

This experiment is designed to study the association between total time delay and the amount of virtual links in one specific AFDX network. The illustration of this experiment is represented in Figure 5-4.

As illustrated in the figure, in this experiment, when one end system is invited as source end system of one virtual link, the other end systems are acted as destination end systems. Three virtual links exist in this experiment which virtual link ids are 300, 400 and 500. Each virtual link contains only one sub-virtual link. In virtual link 300, ES 8 is the source end system while ES 11, ES5 and ES 3 are set as destination end systems. In virtual link 400, ES 14 is the source end system while ES 8, ES 11 and ES 3 are destination end systems. In virtual link 500, ES 3 is the source end system while ES 5, ES 8 and ES 11 are destination end systems. Virtual link 300, 400 and 500 are represented as the solid green

line, solid red line and solid blue line respectively. The $L_{max}$ values of these three virtual links are 200 bytes and the BAG values are 32ms. In this experiment, the amount of virtual links in the AFDX network is set as the variable. At the first execution, virtual link 400 is the only executing virtual link. Virtual link 400 and 500 will be run together in the second execution. At the third execution, all these three virtual links will be executed. Experiment 4 will record the transmission time of FACADE platform on ES 14 and the reception time of FACADE platform on ES 8. These two values are utilised to calculate the total time delay between these two end systems. As the virtual links adding, the experiment will be executed three times while three groups of data will be gained. Those data will then be analysed to detect the association between the amount virtual links in one specific AFDX network and total time delay.

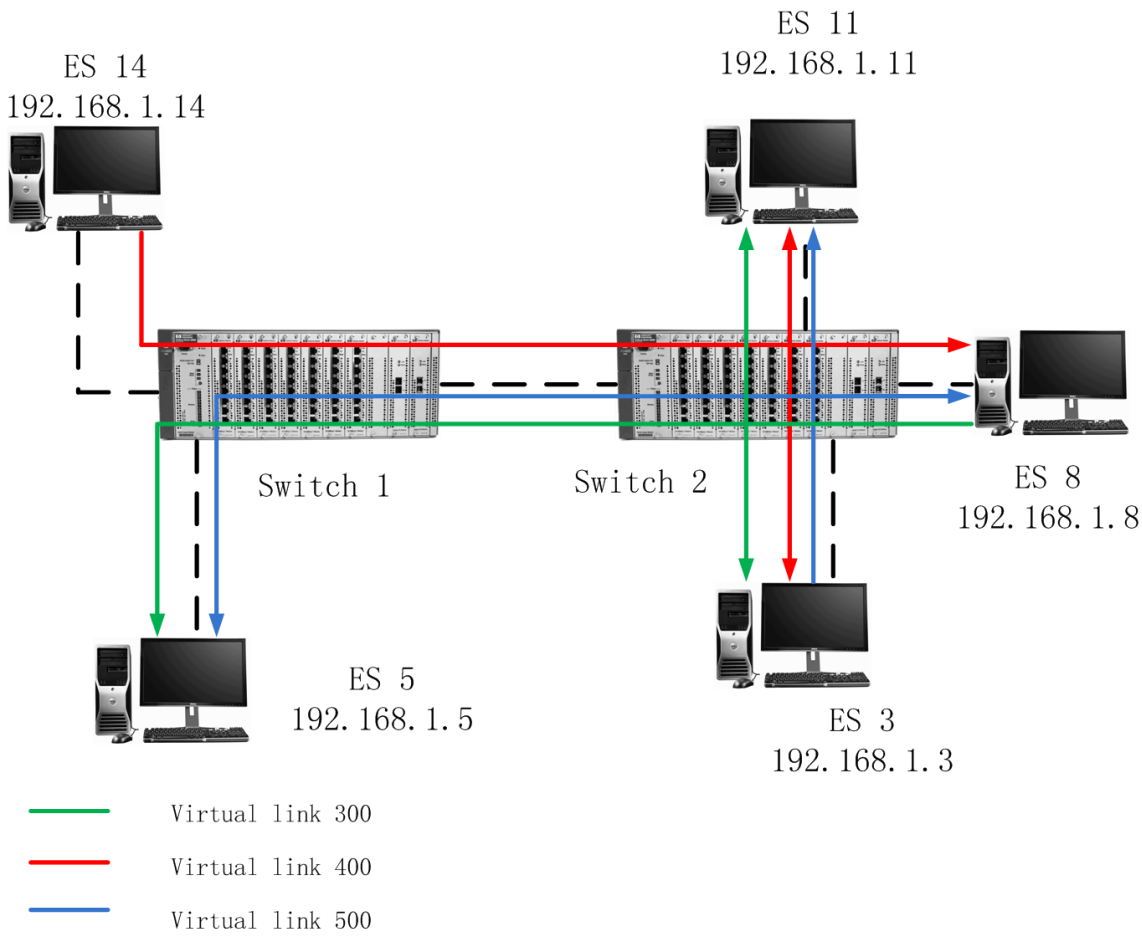


**Figure 5-4 Diagram of Experiment 4**

The total time delay of experiment 4 is represented in Table 5-4.

**Table 5-4 Total Delay Time of Experiment 4**

| Scenario | 1 VL | 2 VL | 3 VL |
|---|---|---|---|
| Time/ms | 128.512 | 128.64 | 128.768 |

### 5.3.5 Experiment 5 (Variable Amount of Traverse Switches)

This experiment is designed to study the association between total time delay and the amount of switches between source end system and destination end system. The illustration of this experiment is represented in Figure 5-5.

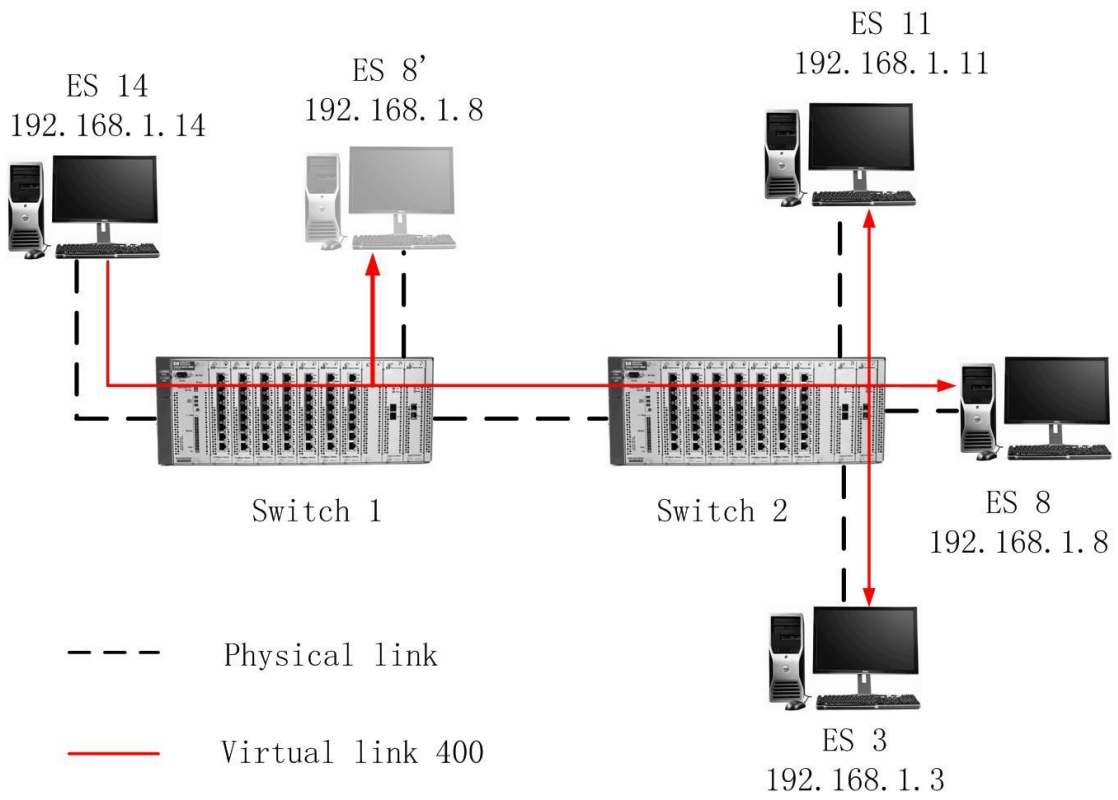


**Figure 5-5 Diagram of Experiment 5**

As shown in the figure, in this experiment, ES 14 is invited as the source end system while other three end systems are destination end systems. Only one virtual link exists in this experiment which virtual link id is 400. Moreover, it has

a unique sub-virtual link. The $L_{max}$ value of this virtual link is 200 bytes and the BAG value is 32ms. In this experiment, the amount of switches between the source end system and destination end systems is set as the variable. In the first execution, ES 8 is connected to switch 2. The amount of switches virtual link 400 traverses is two. In the second execution, ES 8 is represented as ES 8' and connected to switch 1. The amount of switches virtual link 400 traverses is one. This experiment will record the transmission time of FACADE platform on ES 14 and the reception time of FACADE platform on ES 8. These two values are utilised to calculate the total time delay between these two end systems. As the amount of switches virtual link 400 traverses changed, the experiment will be executed twice while two groups of data will be gained. Those data will then be analysed to detect the association between the amount of switches one specific virtual link traverses and total time delay.

The total time delay of experiment 5 is represented in Table 5-5.

**Table 5-5 Total Time Delay of Experiment 5**

| Scenario | Same Switch | Different Switch |
|---|---|---|
| Time/ms | 128.32 | 128.512 |

## 5.4 Experiment States

As represented before, five experiments had been designed. Then these experiments would be executed. In the experiments, avionics applications were utilised to send AFDX data. The AFDX data were transferred more than three thousand times in every single experiment. For each experiment, the data were sent by avionics application simulation platform at the same frequency. Moreover, the data sizes were same. During the experiments, several issues occurred. Some experiments had not been executed as anticipation. The experiments states will be elaborated below.

Experiment 1 set the $L_{max}$ as the variable which values from 100 bytes to 700 bytes, increasing 100 bytes each time. The experiment only succeeded when the $L_{max}$ equals to 200 bytes. The FACADE platform crashed when $L_{max}$ equals

to other values. Due to the time shortage, the author failed to fix this issue. Then the author tried to narrow the values of $L_{max}$ to execute the experiment. After testing numerous times, the experiment 1 was taken when the $L_{max}$ values equal to 200 and 220 bytes. Only two groups of data were obtained instead of seven.

In experiment 2, BAG was set as the variable which values from 2 ms to 64 ms, increasing continuously. The experiment successfully gained data except when BAG was 2 ms. When BAG equals to 2 ms, more than 40% of the total time delay values were negative values and invalid. The author considers that although the time synchronization has been executed each time before the experiment, the time accuracy still can't support to detect the small time interval. Because the computers applied in this experiment are too outmoded. After completed the experiment 2, five groups of data were gained instead of six.

In experiment 3, the amount of destination end systems in one particular virtual link was set as the variable. In this experiment, three different virtual links were invited which would be utilised respectively. The experiment was executed successfully when virtual link 400 was utilised to transfer data as well as virtual link 410. However, simulation platform crashed when virtual link 420 was utilised. The author fell to fix this issue due to the lack of time. In experiment 3, two groups of data were obtained instead of three.

In experiment 4, the amount of virtual links in the AFDX network was set as the variable. In this experiment, three virtual links were invited. The virtual links would be utilised one more than the previous experiment each time. Experiment 4 was completed successfully while no issue happened. Three groups of data were obtained successfully as expected.

In experiment 5, the amount of switches one specific virtual link traverses was set as the variable. In this experiment, ES 8 will firstly be connected to the different switch from source end system ES 14 and then connected to the same switch with ES 14 for the second execution. Experiment 5 was completed successfully while no issue happened. Two groups of data were obtained successfully as expected.

## 5.5 Experiment Data Analysis

After obtaining the experiment data, the mean of each group was calculated each experiment separately. Although the values of the mean are different from group to group, it still can't confirm the variable in each experiment initiate this change. Then the author decides to utilise the student's t-test to detect this issue. Student's t-test is a statistical hypothesis test which can be used to determine if two sets of data are significantly different from each other. To use this test, the groups of data should follow the normal distribution [64]. So the data obtained from experiments should be tested if they follow the normal distribution first.

### 5.5.1 Normal Distribution Test

The test result of data from experiment 1 is illustrated in Figure 5-6.

*NormalityTest*

*Lilliefors*

| | DF | Statistic | p-value | Decision at level(5%) |
|---|---|---|---|---|
| Delay(s) | 3076 | 0.04821 | 3.21038E-18 | Reject normality |
| | 3076 | 0.06209 | 5.34996E-31 | Reject normality |

Delay(s): At the 0.05 level, the data was not significantly drawn from a normally distributed population.
Delay(s): At the 0.05 level, the data was not significantly drawn from a normally distributed population.

**Figure 5-6 Normality Test Result of Experiment 1**

In this experiment, two groups of data have been exchanged successfully. Each group has 3076 delay data. The first group is under the $L_{max}$ value equals to 200 bytes while the other group is under the $L_{max}$ value equals to 220 bytes. After the Normality test by applied the Lilliefors approach, the p-value is 3.21038 $E^{-18}$ when $L_{max}$ value is 200 bytes. When $L_{max}$ value equals to 220 bytes, the p-value is 5.34996 $E^{-31}$. Both p-value values are extremely less than 0.05 which means both groups of data do not follow the normal distribution.

The test result of data from experiment 2 is illustrated in Figure 5-7.

*NormalityTest*

*Lilliefors*

| | DF | Statistic | p-value | Decision at level(5%) |
|---|---|---|---|---|
| Delay | 3056 | 0.13906 | 1.10339E-163 | Reject normality |
| | 3056 | 0.05858 | 2.91432E-27 | Reject normality |
| | 3056 | 0.07426 | 9.10427E-45 | Reject normality |
| | 3056 | 0.0471 | 3.13867E-17 | Reject normality |
| | 3056 | 0.0343 | 8.04462E-9 | Reject normality |

Delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.
Delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.
Delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.
Delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.
Delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.

**Figure 5-7 Normality Test Result of Experiment 2**

In this experiment, five groups of data have been exchanged successfully. Each group has 3056 delay data. The first group is under the BAG value equals to 4 ms while the following groups are under the BAG values equal from 8 ms, 16 ms, 32 ms, 64 ms. After the Normality test by applied the Lilliefors approach, the p-value is 1.10339 $E^{-163}$ when the BAG value is 4 ms. When the BAG values equal from 8 ms, 16 ms, 32 ms and 64 ms, the p-value values are 2.191432 $E^{-27}$, 9.10427 $E^{-45}$, 3.13867 $E^{-17}$ and 8.04462 $E^{-9}$. All the p-value values are less than 0.05 which indicates all the groups of data do not follow the normal distribution.

The test result of data from experiment 3 is illustrated in Figure 5-8.

*NormalityTest*

*Lilliefors*

| | DF | Statistic | p-value | Decision at level(5%) |
|---|---|---|---|---|
| Delay(s) | 3200 | 0.0522 | 1.97219E-22 | Reject normality |
| | 3200 | 0.06941 | 9.46597E-41 | Reject normality |

Delay(s): At the 0.05 level, the data was not significantly drawn from a normally distributed population.
Delay(s): At the 0.05 level, the data was not significantly drawn from a normally distributed population.

**Figure 5-8 Normality Test Result of Experiment 3**

In this experiment, two groups of data have been exchanged successfully. Each group has 3200 delay data. The first group has three destination end systems in one virtual link while the following group has six destination end systems in one virtual link. After the Normality test by applied the Lilliefors approach, the p-

value is 1.97219 $E^{-22}$ when the virtual link has three destination end systems. When the virtual link has six destination end systems, the normality test p-value equals to 9.46597 $E^{-41}$. Both p-value values are extremely less than 0.05 which indicates both groups of data do not follow the normal distribution.

The test result of data from experiment 4 is illustrated in Figure 5-9.

*NormalityTest*

*Lilliefors*

| | DF | Statistic | p-value | Decision at level(5%) |
|---|---|---|---|---|
| delay | 3300 | 0.07021 | 4.52878E-43 | Reject normality |
| | 3300 | 0.05182 | 8.42774E-23 | Reject normality |
| | 3300 | 0.04346 | 9.65838E-16 | Reject normality |

delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.
delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.
delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.

**Figure 5-9 Normality Test Result of Experiment 4**

In this experiment, three groups of data have been exchanged successfully. Each group has 3300 delay data. The first group has only one virtual link executing in the whole AFDX network while the second group and third group have two and three virtual links respectively. After the Normality test by applied the Lilliefors approach, the first group gets a p-value which is 4.52878 $E^{-43}$. At the same time, the second group and third group have p-values which are 8.42774 $E^{-23}$ and 9.65838 $E^{-16}$. All the p-value values are extremely less than 0.05 which indicates all groups of data do not follow the normal distribution.

The test result of data from experiment 5 is illustrated in Figure 5-10.

*NormalityTest*

*Lilliefors*

| | DF | Statistic | p-value | Decision at level(5%) |
|---|---|---|---|---|
| delay | 3300 | 0.04177 | 1.7837E-14 | Reject normality |
| | 3300 | 0.03426 | 1.57139E-9 | Reject normality |

delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.
delay: At the 0.05 level, the data was not significantly drawn from a normally distributed population.

**Figure 5-10 Normality Test Result of Experiment 5**

In this experiment, two groups of data have been exchanged successfully. Each group has 3300 delay data. The first group has the target destination end system connect to the different switch with the source end system. The second group has the target destination end system connect to the same switch with the source end system. After the Normality test by applied the Lilliefors approach, the first group gets a p-value which is 1.57139 $E^{-9}$. The second group has a p-value which is 8.42774 $E^{-23}$. Both p-value values are extremely less than 0.05 which indicates both groups of data do not follow the normal distribution.

The results obtained from normality test show that all the groups of data do not follow the normal distribution respectively. Since then all those can't be tested by student's T-test.

### 5.5.2 Experiment Data Analysis

As mentioned in the last section, the student's t-test can't be applied to analyse the data obtained from experiments. Then the scatter charts and box charts of each experiment are invited for data analysis.

The scatter chart of experiment 1 is represented in Figure 5-11.

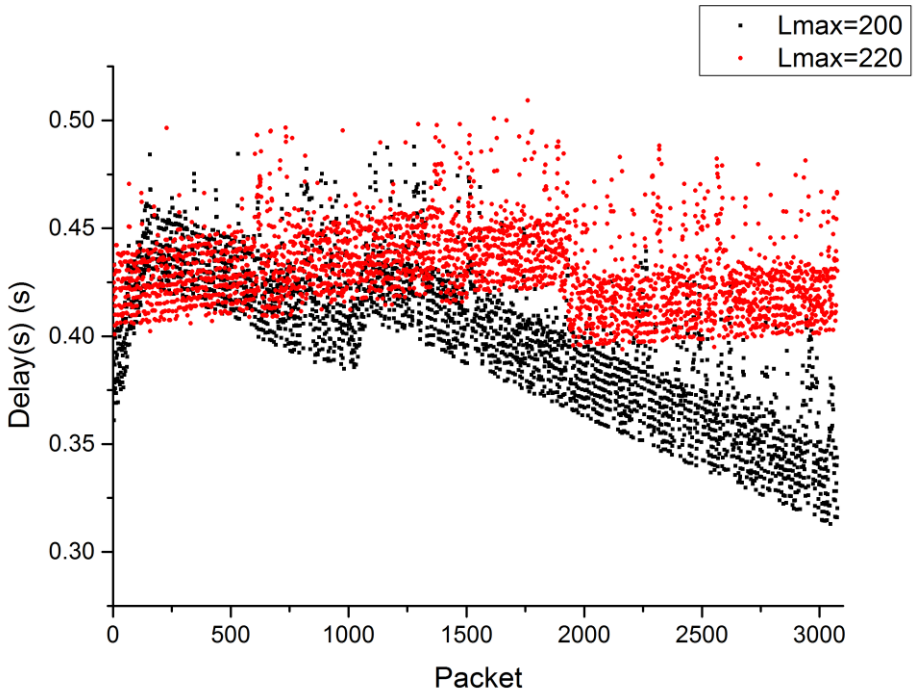


**Figure 5-11 Scatter Chart of Experiment 1**

In this figure, the horizontal axis is the number of AFDX data while the vertical axis represents the values of total time delay, unit in second. The black scatters in the figure represent the total time delay values when the $L_{max}$ equals to 200 bytes while the red scatters illustrate the total time delay values when the $L_{max}$ equals to 220 bytes. As can be seen from this scatter chart, when the $L_{max}$ equals to 200 bytes, the total time delay rose sharply at first. After they reach the peak at around 200, the values decline steadily. Then the values rose markedly to another height at around 1100 and reduce steadily again since then. On the other hand, when the $L_{max}$ equals to 220 bytes, the performance of total time delay rose steadily till around 1900. Then at this point, the values of delay decrease sharply and rose steadily again since then. The trends of these two groups of data indicate that when $L_{max}$ equals to 200 bytes, the values of total time delay decline as the AFDX data transmission. On the contrary, the total time delay when $L_{max}$ equals to 220 bytes raise smoothly.

The box charts of these two groups of data are illustrated in Figure 5-12.

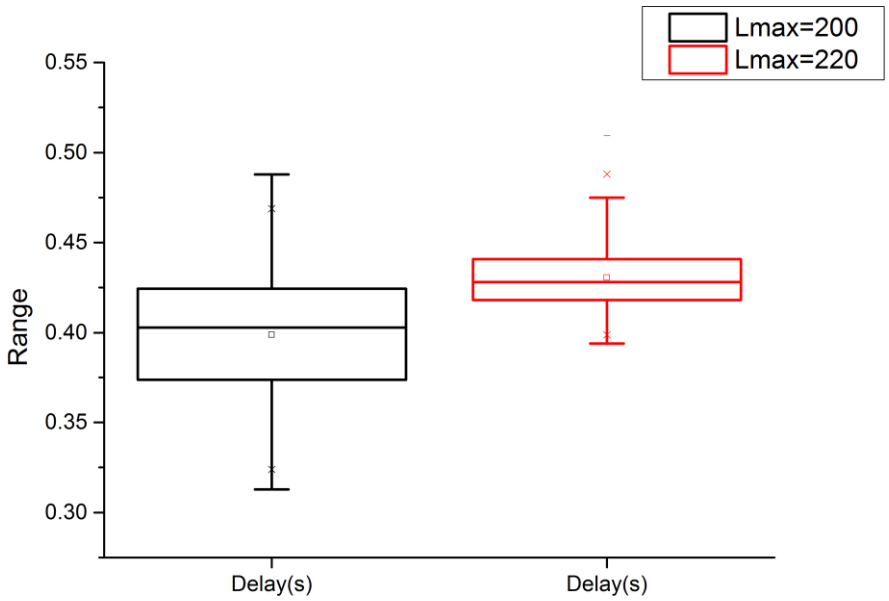


**Figure 5-12 Box Chart of Experiment 1**

In this figure, the data of experiment 1 are represented as box chart. The horizontal axis is the values of total time delay, unit in second. The black box represents the data when the $L_{max}$ is 200 bytes while the red box shows the data when the $L_{max}$ is 220 bytes. As shown in this figure, the mean when $L_{max}$ is 200 bytes is less than the mean when $L_{max}$ is 220 bytes. It indicates when $L_{max}$ is 200 bytes, it gets a smaller total time delay than $L_{max}$ is 220 bytes.

The means of data from experiment 1 are illustrated in Figure 5-13.

*Descriptive Statistics*

| | N total | Mean | Minimum | Median | Maximum |
|---|---|---|---|---|---|
| Delay(s) | 3076 | 0.39889 | 0.31286 | 0.40291 | 0.48783 |
| | 3076 | 0.43059 | 0.39395 | 0.42812 | 0.50921 |

**Figure 5-13 Statistics Data of Experiment 1**

In Figure 5-13, the first row represents the total time delay of experiment 1 when $L_{max}$ equals to 200 bytes. The second row shows the total time delay when $L_{max}$ equals to 220 bytes. As shown in Figure 5-13, when $L_{max}$ equal to

200 bytes and 220 bytes, the mean of total time delay are 398.89 ms and 430.59 ms. The maximum total time delays are 487.83 ms and 509.21 ms when $L_{max}$ are 200 and 220 bytes respectively. Both total time delays are much larger than the delay calculus before which are 128.512 ms and 128.550 ms respectively.

**Table 5-6 Comparison between Calculus Delay and Experiment Delay (Experiment 1)**

| $L_{max}$/bytes | Calculus Delay/ms | Mean of Total Time Delay/ms | Multiples (Mean Compares to Calculus) | Maximum of Total Time Delay/ms | Multiples (Maximum Compares to Calculus) |
|---|---|---|---|---|---|
| 200 | 128.512 | 398.89 | 3.10 | 487.83 | 3.80 |
| 220 | 128.550 | 430.59 | 3.35 | 509.21 | 3.96 |

The scatter chart of experiment 2 is represented in Figure 5-14.

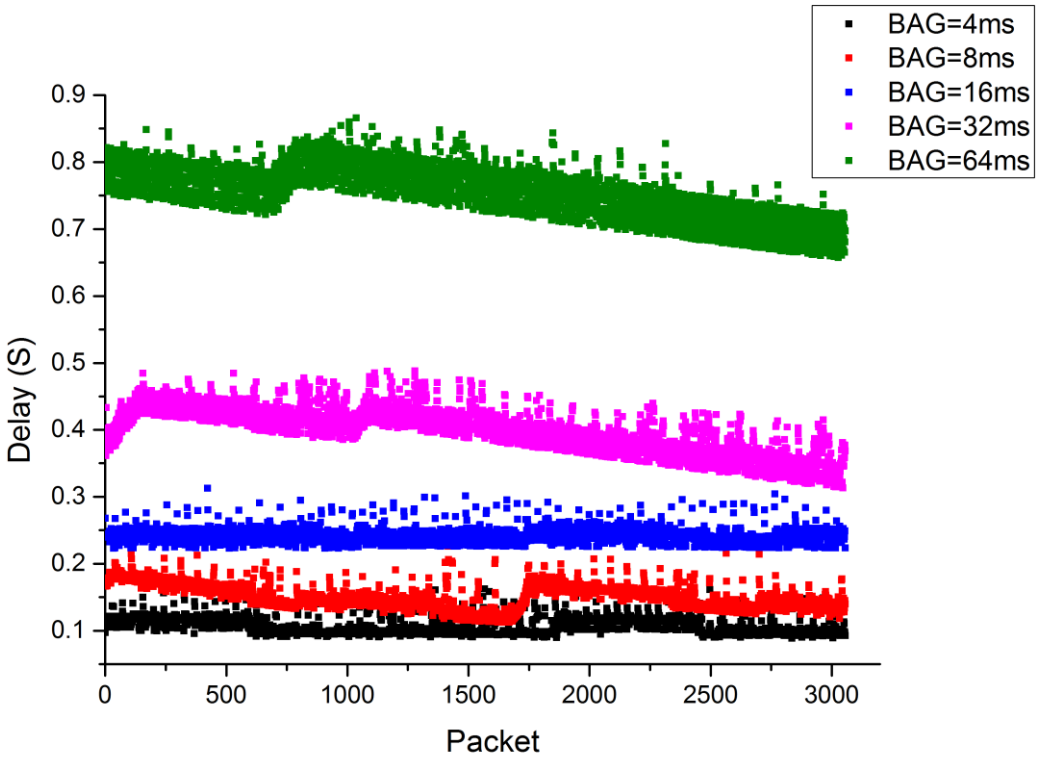


**Figure 5-14 Scatter Chart of Experiment 2**

In this figure, the horizontal axis is the number of AFDX data while the vertical axis represents the values of total time delay, unit in second. The green scatters represent the total time delay values when the BAG is 64 ms. The purple scatters illustrate the total time delay values when the BAG is 32 ms. The blue, red and black scatters indicate the total time delay values when the BAG are 16 ms, 8 ms and 4 ms respectively. When the BAG is 64 ms, the delay values declined from the beginning to around 800, then the delay rose sharply around 900 and deduced steadily again. When the BAG is 32 ms, the delay values increase sharply from the start to around 200. Then the delay decreased smoothly since then. When the BAG is 16 ms, the delay values were smooth than the previous. The values kept level and smooth in the whole data exchange duration. When the BAG is 8 ms, the delay values decreased steadily from the beginning to around 1700 and rose sharply around 1800. After that, the delay declined since then. When the BAG is 4 ms, the delay values kept smoothly from the beginning to around 650 and reduced. Then the delay went straight till around 1900 and increased. After that, the delay continued smoothly till around 2500 and reduced again. Finally, the delay went straight till the end.

The box charts of these five groups of data are represented in Figure 5-15.

In this figure, the data of experiment 2 are represented as box chart. The horizontal axis is the values of total time delay, unit in second. The green one represents the data when the BAG is 64 ms while the purple one illustrates the data when the BAG is 32 ms. Meanwhile, the blue, red and black box chart show the data of 16 ms, 8 ms and 4 ms respectively.

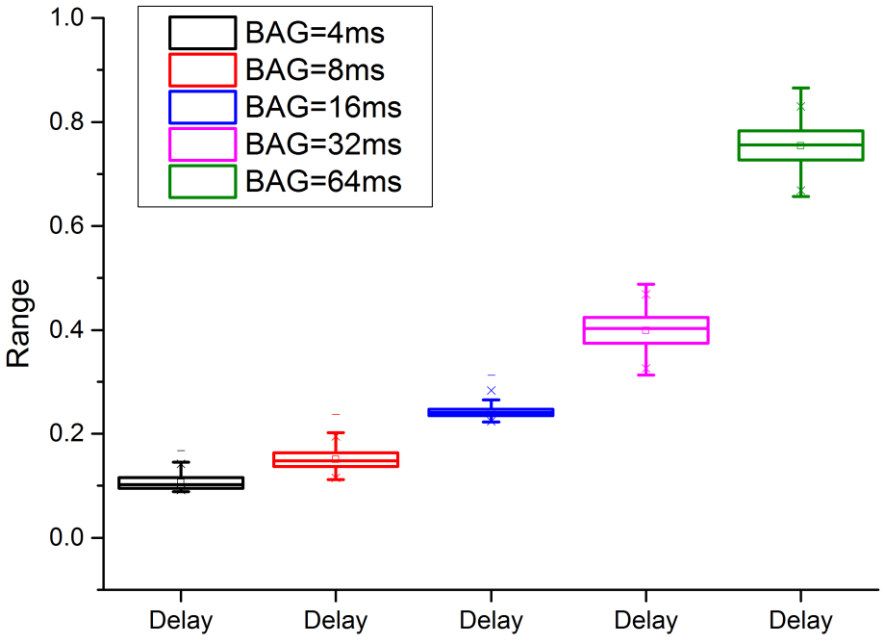


**Figure 5-15 Box Chart of Experiment 2**

The statistics data from experiment 2 are illustrated in Figure 5-16.

*Descriptive Statistics*

| | N total | Mean | Minimum | Median | Maximum |
|---|---|---|---|---|---|
| Delay | 3056 | 0.10614 | 0.08865 | 0.10217 | 0.16754 |
| | 3056 | 0.15023 | 0.11215 | 0.14769 | 0.23696 |
| | 3056 | 0.24175 | 0.22268 | 0.24099 | 0.31216 |
| | 3056 | 0.3993 | 0.31286 | 0.40321 | 0.48783 |
| | 3056 | 0.75387 | 0.65684 | 0.75602 | 0.86544 |

**Figure 5-16 Statistics Data of Experiment 2**

In Figure 5-16, five rows of data are shown. All these rows represent the statistics data of experiment 2 when BAG equal to 4 ms, 8 ms, 16 ms, 32 ms and 64 ms from the first row to the fifth row respectively. As shown in Figure 5-16, when BAG equal from 4 ms to 64 ms, the means of total time delay are 106.14 ms 150.23 ms, 241.75 ms, 399.30ms and 753.87 ms respectively.

Meanwhile, the maxima of total time delay are 167.54 ms, 236.96 ms, 312.16 ms, 487.83 ms and 865.44 ms. All the total time delay are much larger than the calculus delay which are 16.512 ms, 32.512 ms, 64.512 ms, 128.512 ms and 256.512 ms.

**Table 5-7 Comparison between Calculus Delay and Experiment Delay (Experiment 2)**

| BAG/ms | Calculus Delay/ms | Mean of Total Time Delay/ms | Multiples (Mean Compares to Calculus) | Maximum of Total Time Delay/ms | Multiples (Maximum Compares to Calculus) |
|---|---|---|---|---|---|
| 4 | 16.512 | 106.14 | 6.43 | 167.54 | 10.15 |
| 8 | 32.512 | 150.32 | 4.62 | 236.96 | 7.29 |
| 16 | 64.512 | 241.75 | 3.75 | 312.16 | 4.84 |
| 32 | 128.512 | 399.30 | 3.11 | 487.83 | 3.80 |
| 64 | 256.512 | 753.87 | 2.94 | 865.44 | 3.37 |

The scatter chart of experiment 3 is represented in Figure 5-17.

In this figure, the horizontal axis is the number of AFDX data while the vertical axis represents the values of total time delay, unit in second. The black scatters represented the total time delay of one sub-virtual link while the red scatters illustrated the total time delay of two sub-virtual links. The distributions of these two data are quite similar. Box chart and statistic data will demonstrate the difference more clearly which will be represented next.

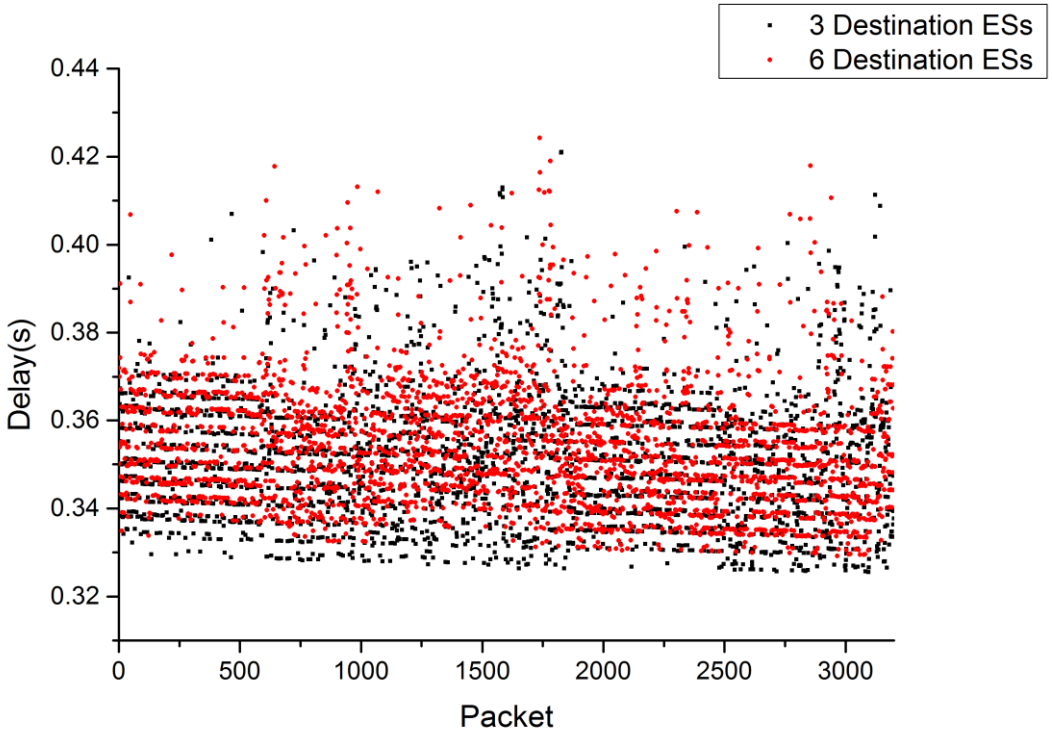


**Figure 5-17 Scatter Chart of Experiment 3**

The box charts of these five groups of data are represented in Figure 5-18.

In this figure, the data of experiment 3 are represented as box charts. The horizontal axis is the values of total time delay, unit in second. It can be seen from this figure that the two sub-virtual links scenario has a larger mean of total time delay while the one sub-virtual link scenario has a smaller mean of delay. The scale of each quartile about these two scenarios is quite similar. And the difference between these two means is quite small.

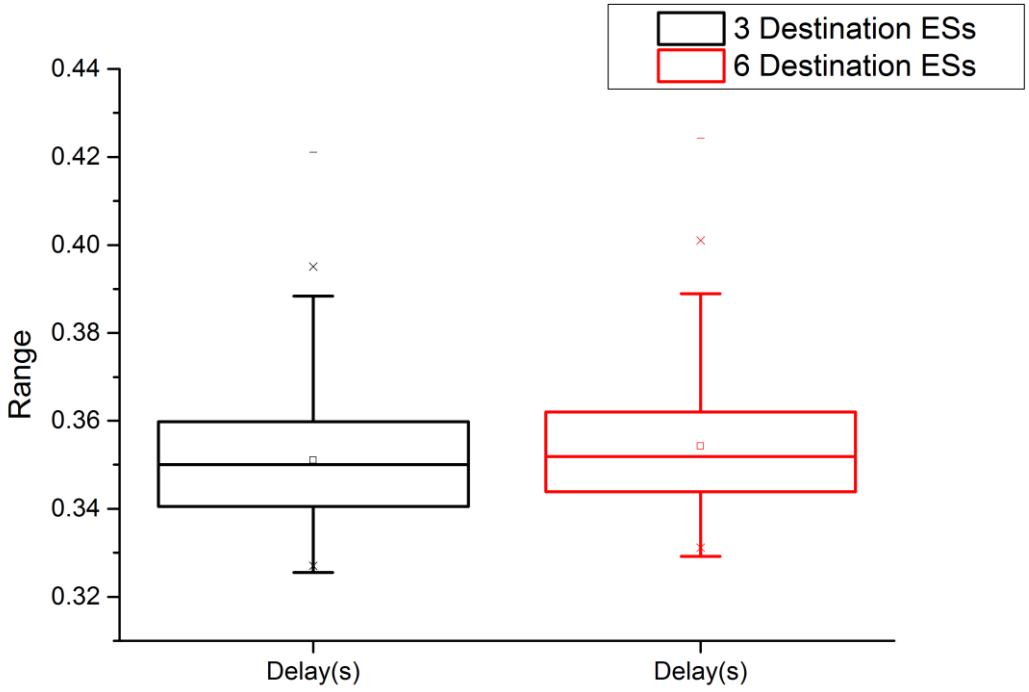


**Figure 5-18 Box Chart of Experiment 3**

The statistics data from experiment 3 are illustrated in Figure 5-19.

*Descriptive Statistics*

| | N total | Mean | Minimum | Median | Maximum |
|---|---|---|---|---|---|
| Delay(s) | 3200 | 0.35104 | 0.32549 | 0.35 | 0.42105 |
| | 3200 | 0.35421 | 0.32921 | 0.35184 | 0.42428 |

**Figure 5-19 Statistics Data of Experiment 3**

In Figure 5-19, the first row represents the statistics data of total time delay when the virtual link has three destination end systems. The second row shows the statistics data of total time delay when the virtual link has six destination end systems. As shown in Figure 5-19, when the amount of destination end systems are three and six respectively, the means of total time delay are 351.04 ms and 354.21 ms. Meanwhile, the maxima of total time delay are 421.05 ms and 424.28 ms when the virtual link has three destination end systems and six

destination end systems. Both total time delays are much larger than the calculus delays which are 128.512 ms and 128.512 ms.

**Table 5-8 Comparison between Calculus Delay and Experiment Delay (Experiment 3)**

| Destination End systems In one Virtual link | Calculus Delay | Mean of Total Time Delay | Multiples (Mean Compares to Calculus) | Maximum of Total Time Delay | Multiples (Maximum Compares to Calculus) |
|---|---|---|---|---|---|
| 3 | 128.512 | 351.04 | 2.73 | 421.05 | 3.28 |
| 6 | 128.512 | 354.21 | 2.76 | 424.28 | 3.30 |

The scatter chart of experiment 4 is represented in Figure 5-20.

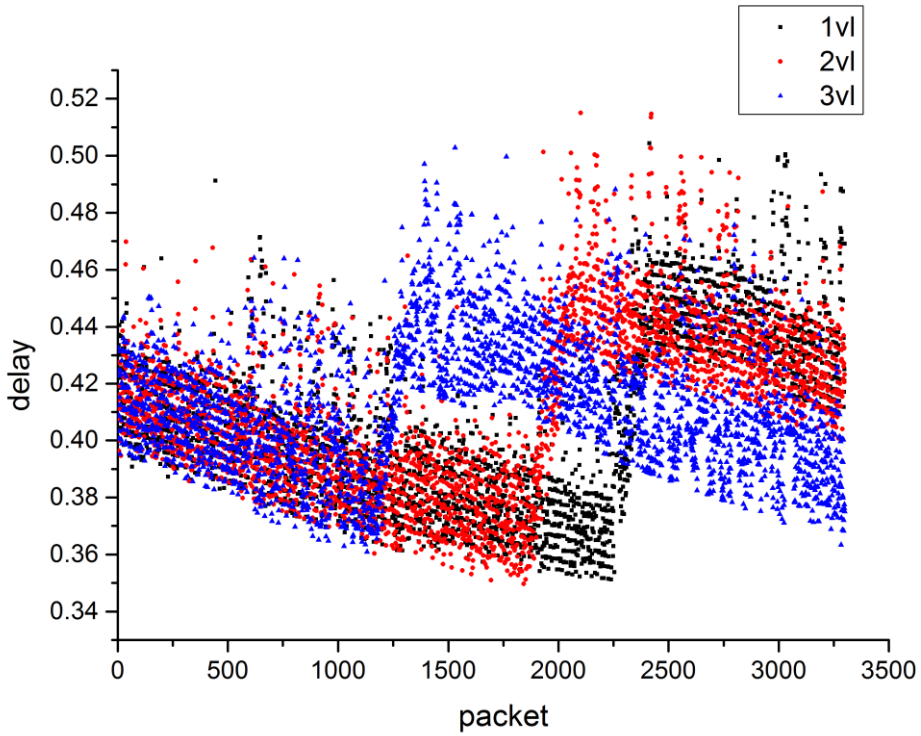


**Figure 5-20 Scatter Chart of Experiment 4**

In this figure, the horizontal axis is the number of AFDX data while the vertical axis represents the values of total time delay, unit in second. The black scatters represented the total time delay values when there is one virtual link in the experiment AFDX network. The red and blue scatters illustrate the total time delay values when there are two and three virtual links in the experiment AFDX network respectively. As can be seen in this figure, when the amount of virtual link is one, the delay reduced steadily from the beginning to around 2250 and then increased sharply till around 2400. Since then the delay declined steadily till the end. The other two scatters have the similar expressions. When virtual link amount is two, the delay reduced steadily from the beginning to around 1850 and then rose dramatically till around 2200. Since then the delay declined steadily till the end. When virtual link amount is three, the delay reduced steadily from the beginning to around 1200 and then increased sharply till around 1400. Since then the delay declined steadily till the end.

The box charts of these three groups of data are represented in Figure 5-21.

In this figure, the data of experiment 4 are represented as box chart. The horizontal axis is the values of total time delay, unit in second. The black one represents the delay data when there is one virtual link in the experiment AFDX network while the red one illustrates the delay data when there are two virtual links. The blue box shows the delay data when there are three virtual links.

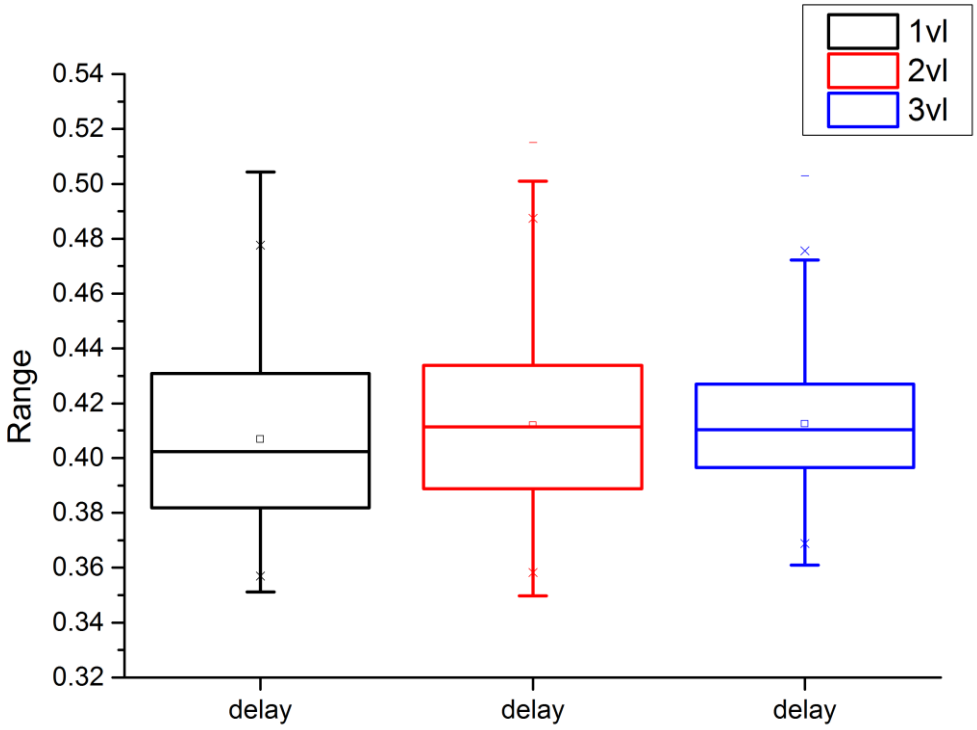


**Figure 5-21 Box Chart of Experiment 4**

The statistics data from experiment 4 are illustrated in Figure 5-22.

*Descriptive Statistics*

| | N total | Mean | Minimum | Median | Maximum |
|---|---|---|---|---|---|
| delay | 3300 | 0.40685 | 0.35119 | 0.40228 | 0.5043 |
| | 3300 | 0.41203 | 0.34969 | 0.41126 | 0.51506 |
| | 3300 | 0.41247 | 0.36092 | 0.41032 | 0.50287 |

**Figure 5-22 Statistics Data of Experiment 4**

In Figure 5-22, three rows of data express the statistics data of total time delay when virtual links in AFDX network are one, two and three respectively. As shown in Figure 5-22, when the amount of virtual links in AFDX network are one, two and three respectively, the means of total time delay are 406.85 ms, 412.03 ms and 412.47 ms. Meanwhile, the maxima of total time delay are 504.3 ms, 515.06 ms and 502.87 ms. However, all the experiment delay are much larger than the calculus delay which are 128.512 ms, 128.64 ms and 128.768 ms.

**Table 5-9 Comparison between Calculus Delay and Experiment Delay (Experiment 4)**

| Amount of Virtual Link | Calculus Delay/ms | Mean of Total Time Delay/ms | Multiples (Mean Compares to Calculus) | Maximum of Total Time Delay/ms | Multiples (Maximum Compares to Calculus) |
|---|---|---|---|---|---|
| 1 | 128.512 | 406.85 | 3.17 | 504.3 | 3.92 |
| 2 | 128.64 | 412.03 | 3.20 | 515.06 | 4.00 |
| 3 | 128.768 | 412.47 | 3.20 | 502.87 | 3.91 |

The scatter chart of experiment 5 is represented in Figure 5-23.

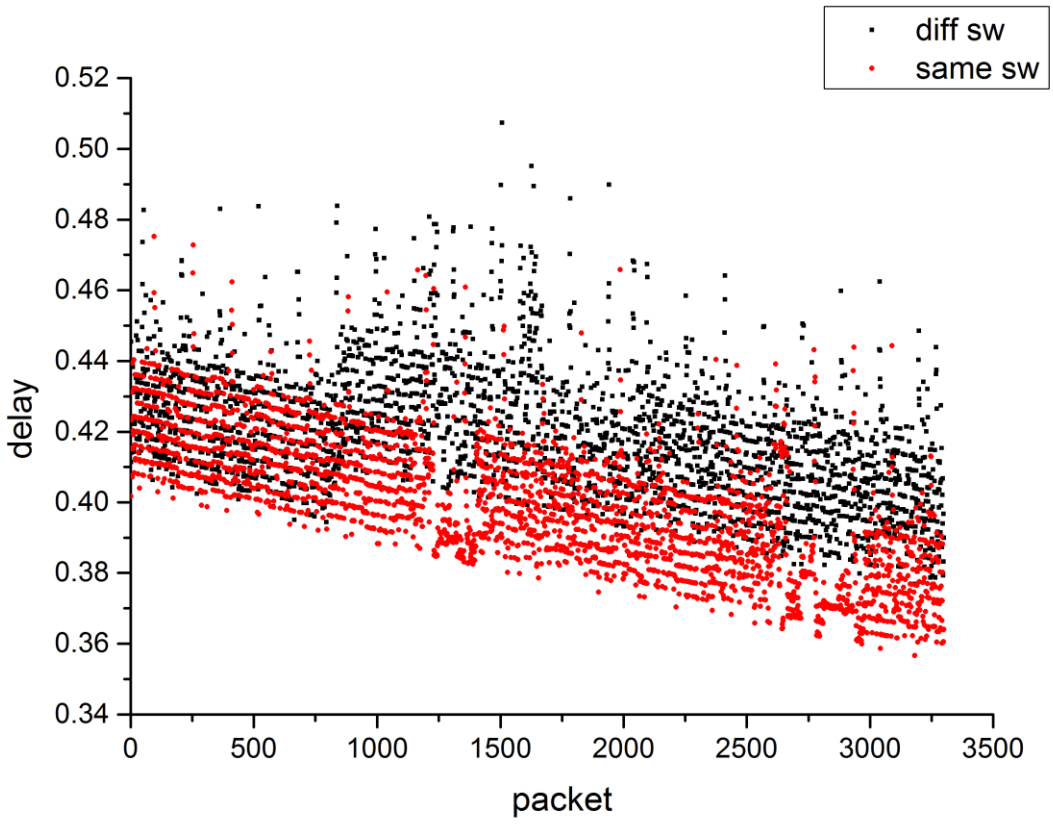


**Figure 5-23 Scatter Chart of Experiment 5**

In this figure, the horizontal axis is the number of AFDX data while the vertical axis represented the values of total time delay, unit in second. The black scatters represented the target destination end systems connected to the different switches from the source end system. The red scatters illustrated the target destination end systems connected to the same switch with the source end system. As can be seen from the figure, when the target destination end system and source end system were connected to the different switches, the values of total time delay declined from the beginning till around 800 and then increased sharply. Since then, most of the values declined steadily till the end. When target destination end system and source end system were connected to the same switch, most total time delay values declined steadily from beginning till the end.

The box charts of these two groups of data are represented in Figure 5-24.

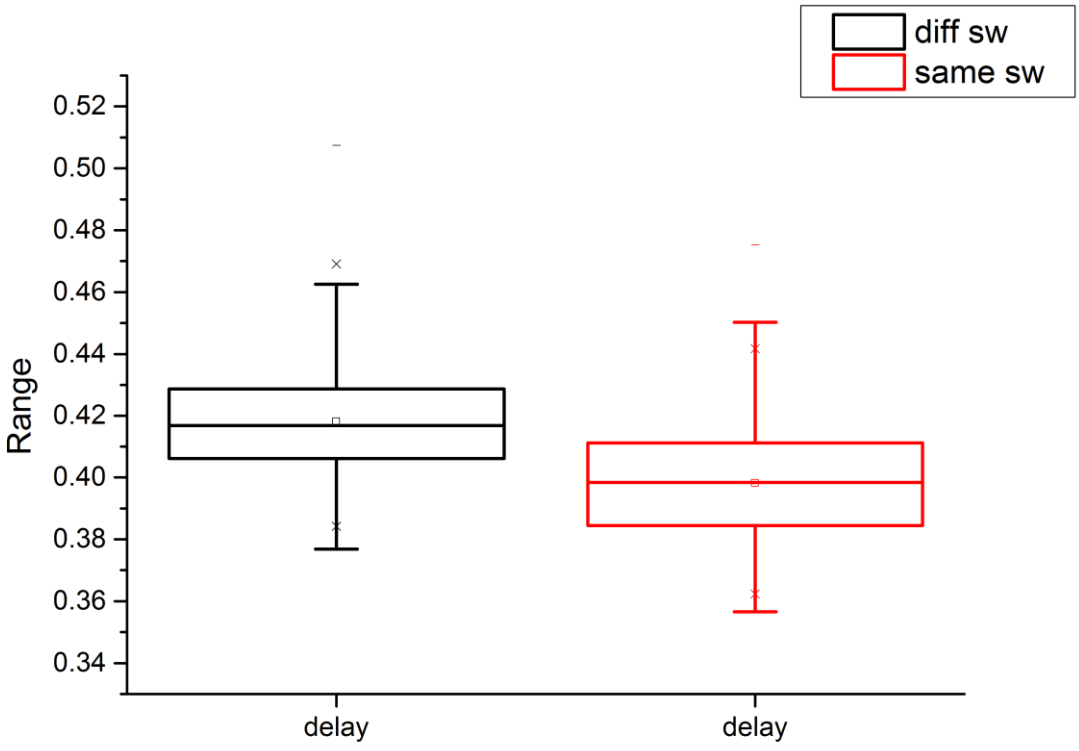


**Figure 5-24 Box Chart of Experiment 5**

In this figure, the data of experiment 5 are represented as box chart. The horizontal axis is the values of total time delay, unit in second. The black box indicates the target destination end system and source end system are connected to the different switches while the red box refers these two end systems are connected to the same switch. It can be seen from the figure that when these two end systems are connected to the same switch, the mean of total time delay is smaller than the mean of delay when these two end systems are connected to the different switches.

The statistics data from experiment 5 are illustrated in Figure 5-25.

*Descriptive Statistics*

| | N total | Mean | Minimum | Median | Maximum |
|---|---|---|---|---|---|
| delay | 3300 | 0.41817 | 0.37684 | 0.41688 | 0.50739 |
| | 3300 | 0.39817 | 0.35661 | 0.39837 | 0.47519 |

**Figure 5-25 Statistics Data of Experiment 5**

In Figure 5-25 two rows of data represents the statistics data of total time delay. The first line shows the delay when two switches between source and destination end systems while the second row expresses the delay when one switch between source and destination end systems. As shown in Figure 5-25, when the amount of switch one virtual links cross is two and one respectively, the means of total time delay are 418.17 ms and 398.17 ms. Meanwhile, the maxima of total time delay are 507.39 ms and 475.19 ms. Both of the total time delays are much larger than the calculus delay which are 128.32 ms and 128.512 ms.

**Table 5-10 Comparison between Calculus Delay and Experiment Delay (Experiment 5)**

| Amount of Switches Virtual Link Cross | Calculus Delay/ms | Mean of Total Time Delay/ms | Multiples (Mean Compares to Calculus) | Maximum of Total Time Delay/ms | Multiples (Maximum Compares to Calculus) |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 2 | 128.512 | 418.17 | 3.25 | 507.39 | 3.95 |
| 1 | 128.32 | 398.17 | 3.10 | 475.19 | 3.70 |

## 5.6 Summary

In this chapter, experiment objectives are introduced first. These experiments are designed to research the associations between total time delay and other variables which are $L_{max}$, BAG, the amount of destination end systems in one specific virtual link, the amount of virtual links in one specific network and the amount of switches between source and destination end systems. Secondly, the experiment environment is represented and the approaches to obtain the total time delay are introduced. At first, the author tries to use command ‘time’ but failed. Then the author applies the NTP service for time synchronization. While the automatic time synchronization is convenient but inaccurate, manual time synchronization is much more accurate. Then the manual time synchronization is adopted in this project. Thirdly, the detailed design five experiments are elaborated respectively. Fourthly, the experiment execution states and data analysis are introduced. Fifthly, the experiment execution states and the issues during the experiments are elaborated. Then, to apply student’s t-test for data analysis, all those data obtained from experiments are tested to detect whether they followed the normal distribution. At last, since those data can’t be analyzed by student’s t-test, scatters chart and box chart are introduced. Meanwhile, statistics data of the total time delay are proposed and compare with the calculus results.

# 6 CONCLUSIONS AND FUTURE WORK

## 6.1 Introduction

At the beginning of the research, the author studies the recently AFDX research related articles. Several approaches have been summarised. The theoretical approaches are mainly utilised to invite the mathematical approaches for obtaining the tighter AFDX network time delay. Simulations methods are used for proving the theoretical approaches. This type of approaches can also provide useful information for AFDX network design. Some researchers invited real-time software to simulate the AFDX network environment for research purpose. Without hardware influence, this type of simulations may miss the impact from hardware and not similar to the realistic environment. On the contrary, the programming software simulation with network hardware is more realistic. This approach simulates the AFDX network data exchange behaviour with both software and hardware is similar to the industrial environment. This programming software implementation platform can also be applied for the education and research purpose.

Then the AFDX time delay model is given by using the Network Calculus. In this section, the whole AFDX network is divided into three main components which are end system, switch and physical propagation. The time delay models of the end system and switch are given to obtain the total time delay. In this part, both (σ,ρ) model and GCRA model are introduced. To get a tighter delay bound, the GCRA model is applied in this project. Then the formula for total time delay is illustrated which then be utilised to gain the calculus results.

In this project, an executable FACADE platform has been enhanced bases on the previous platform. The former students have successfully developed a flexible platform. However, this platform can only show the data exchanging between source end system and destination end systems. It can’t prove whether the data transmitted by the FACADE platform is valid or not. Moreover, this simulation platform can’t prove whether there is any missing data. The author has successfully modified the previous simulation platform. This

modification platform can be utilised to detect whether the data is invalid after received by destination end system. Missing AFDX data can also be identified. Moreover, this modified simulation platform is suitable for experiment purpose which can be utilised to obtain the total time delay. Users can modify the architecture of the AFDX network base on their experiment objectives.

To study the association between total time delay and other variables which are $L_{max}$, BAG, the amount of destination end systems in one specific virtual link, the number of virtual links in one particular network and the amount of switches between source and destination end systems. After that, the experiment environment is represented and the approaches to obtain the total time delay are introduced. At first, the author tries to use command 'time' but failed. Then the NTP service is proposed for the time synchronization. After the trail, the manual time synchronization approach is applied in this project due to the time accuracy. Next, five experiments are designed. Then these five experiments are elaborated respectively.

After that, the experiment execution states and data analysis are introduced. Firstly, the experiment execution states and the issues during the experiments are listed. Secondly, to apply student's t-test for data analysis, all those data obtained are tested whether they follow the normal distribution. Thirdly, since the student's t-test can't be proposed in this project, the scatters chart, box chart and statistics data are invited to analyse the obtained data. Meanwhile, the calculus results and experiment results of total time delay are compared, and the conclusions are represented.

## 6.2 Conclusions

In this section, the conclusions obtained from the project are summarised as follows:

Each AFDX data packet in this project will be divided into four fragmented messages for further transmission no matter $L_{max}$ is 200 or 220 bytes. Since larger divided AFDX data needs more time to transfer on the network. $L_{max}$ equals to 200 bytes has a smaller means of total time delay than $L_{max}$ equals to

220 bytes. This suggests the designer to consider the influence of $L_{max}$ to the total time delay of the AFDX network according to the requirement of avionics application.

The larger the BAG is, the greater the total delay it obtains. The author considers that the larger BAG leads to longer transmission duration. Although regulator regulates the unregulated flows, the extra waiting time it brings to the data transmission constrains the utility of each virtual link.

The number of destination end systems in one virtual link may affect the total time delay of this virtual link. Results from this project can't validate. More studies should be conducted.

The total time delay of one particular virtual link may be affected by other virtual links in the same AFDX network. The more virtual links across the switch which the target virtual link through, the larger a total time delay it seems to be. More studies should be conducted for verifying.

The more switches between source and destination end systems, the larger a total time delay it will be. More switches introduced more scheduler delay and hardware process time which leads to larger total time delay.

The means and maxima of all the groups of data from experiments are obtained. After comparing those data with the experiment data, it can be seen that the calculus results from AFDX time delay model are much smaller than the delay from experiments. The mean of total time delay could be 6.43 times of calculus one. The maximum of total time delay could be 10.15 times of the calculus one. The author considers that due to the background system process running in the operating system, the system resource can be occupied. When FACADE platform is executing, the execution time could be extremely higher than it needs.

## 6.3 Future Work

Platform Error

During the experiment, when the author tried to modify the $L_{max}$ values of the FACADE platform, the platform crashed after it transferred dozens of AFDX data. Due to this reason, the experiment 1 can only obtain two groups of data. The same situation happened when more destination end systems were added to the same sub-virtual link which makes the experiment 3 can't be executed as it predefined. Because of the time shortage, the author failed to fix these issues. This problem should be handled for the robust of FACADE platform in the future.

Platform Efficiency

During the experiments, the author had done numerous experiments and a large number of total time delay values were obtained. The experiment data are quite high than the calculus delays. The author considered that the delay values can be smaller by reducing the background system process and optimizing the simulation platform process. To achieve this, the operating system applied in this project may need to be optimized by eliminating the unneeded system processes. Moreover, the time consumption among these modules and sub modules can be detected for further platform optimization.

More Accurate Time Obtained Method

According to experiment 2, when BAG is 2 ms, more than 40% obtained transmission time values are negative. The author considers that although the manual time synchronization approach has been proposed to gain accurate time among these experiment computers, it still not accurate enough for obtaining a smaller time value. That is why the author chose BAG equals to 32 ms for other experiments.

The following task could focus on the time deviations of each client. As long as the ranges of time deviations are obtained, the experiment delay could be more accurate.

Another possible way is to send packet by destination end system once this packet arrives and then to record both of total time delays. The average of these total time delays may be the exact total time delay.

# REFERENCES

[1] Airlines Electronic Engineering Committee. (2005). ARINC Specification 664P7: Aircraft Data Network, Part 7–Avionics Full Duplex Switched Ethernet (AFDX) Network. Aeronautical Radio Inc.

[2] Moir, I., Seabridge, A., & Jukes, M. (2013). Civil avionics systems. John Wiley & Sons.

[3] Airlines Electronic Engineering Committee. (2003). ARINC 653—Avionics Application Software Standard Interface.

[4] Tawk, M., Zhu, G., Savaria, Y., Liu, X., Li, J., & Hu, F. (2011, October). A tight end-to-end delay bound and scheduling optimization of an avionics AFDX network. In Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th (pp. 7B3-1). IEEE.

[5] Cruz, R. L. (1991). A calculus for network delay. I. Network elements in isolation. Information Theory, IEEE Transactions on, 37(1), 114-131.

[6] Cruz, R. L. (1991). A calculus for network delay. II. Network elements in isolation. Information Theory, IEEE Transactions on, 37(1), 132-141.

[7] Fraboul, C., & Frances, F. (2002). Applicability of Network Calculus to the AFDX. contract report PBAR-JD-728.0821.

[8] Airlines Electronic Engineering Committee. (2002). ARINC Specification 664P1: Aircraft Data Network, Part 1–Systems Concepts and Overview. Aeronautical Radio Inc.

[9] Airlines Electronic Engineering Committee. (2002). ARINC Specification 664P2: Aircraft Data Network, Part 2–Ethernet Physical and Data Link Layer. Aeronautical Radio Inc.

[10] Martin, S., & Minet, P. (2006, April). Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. In Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International (pp. 8-pp). IEEE.

[11] Bauer, H., Scharbarg, J. L., & Fraboul, C. (2009, September). Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on (pp. 1-8). IEEE.

[12] Frances, F., Fraboul, C., & Grieu, J. (2006). Using network calculus to optimize the AFDX network.

[13] Li, X., Scharbarg, J. L., & Fraboul, C. (2010, September). Improving end-to-end delay upper bounds on an AFDX network by integrating offsets in worst-case analysis. In Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on (pp. 1-8). IEEE.

[14] Bauer, H., Scharbarg, J. L., & Fraboul, C. (2010). Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach.Industrial Informatics, IEEE Transactions on, 6(4), 521-533.

[15] Diaz, J. L., Lopez, J. M., Garcia, M., Campos, A. M., Kim, K., & Bello, L. L. (2004, December). Pessimism in the stochastic analysis of real-time systems: Concept and applications. In Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International (pp. 197-207). IEEE.

[16] Scharbarg, J. L., Ridouard, F., & Fraboul, C. (2009). A probabilistic analysis of end-to-end delays on an AFDX avionic network. Industrial Informatics, IEEE Transactions on, 5(1), 38-49.

[17] George, L., Rivierre, N., & Spuri, M. (1996). Preemptive and non-preemptive real-time uniprocessor scheduling.

[18] Davis, R. I., Burns, A., Bril, R. J., & Lukkien, J. J. (2007). Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems, 35(3), 239-272.

[19] Andersson, B., & Tovar, E. (2009, July). The Utilization Bound of Non-Preemptive Rate-Monotonic Scheduling in Controller Area Networks is 25 percent. In SIES 2009. IEEE Symposium on Industrial Embedded Systems.

[20] Tindell, K., Burns, A., & Wellings, A. J. (1995). Calculating controller area network (CAN) message response times. Control Engineering Practice, 3(8), 1163-1169.

[21] Liu, C. L., & Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 20(1), 46-61.

[22] Li, D. J., Zhang, J. D., & Liu, B. (2010, June). Periodic message-based modeling and performance analysis of AFDX. In Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on (pp. 162-166). IEEE.

[23] Jiandong, Z., Dujuan, L., & Yong, W. (2010, May). Modelling and performance analysis of AFDX based on Petri net. In Future Computer and Communication (ICFCC), 2010 2nd International Conference on (Vol. 2, pp. V2-566). IEEE.

[24] Alur, R., & Dill, D. L. (1994). A theory of timed automata. Theoretical computer science, 126(2), 183-235.

[25] Charara, H., Scharbarg, J. L., Ermont, J., & Fraboul, C. (2006). Methods for bounding end-to-end delays on an AFDX network. In Real-Time Systems, 2006. 18th Euromicro Conference on (pp. 10-pp). IEEE.

[26] Dong, S., Xingxing, Z., Lina, D., & Qiong, H. (2010, October). The design and implementation of the AFDX network simulation system. In Multimedia Technology (ICMT), 2010 International Conference on (pp. 1-4). IEEE.

[27] Lina, D., Dong, S., Xingxing, Z., & Qiong, H. (2010, October). The research of AFDX system simulation model. In Multimedia Technology (ICMT), 2010 International Conference on (pp. 1-4). IEEE.

[28] Charara, H., & Fraboul, C. (2005, July). Modelling and simulation of an avionics full duplex switched ethernet. In Telecommunications, 2005. advanced industrial conference on telecommunications/service assurance with partial and

intermittent resources conference/e-learning on telecommunications workshop. aict/sapir/elete 2005. proceedings (pp. 207-212). IEEE.

[29] Véran, M., & Potier, D. (1984). QNAP 2: A portable environment for queueing systems modelling.

[30] Bozga, M., Graf, S., Ober, I., Ober, I., & Sifakis, J. (2004). The IF toolset. InFormal Methods for the Design of Real-Time Systems (pp. 237-267). Springer Berlin Heidelberg.

[31] Gössler, G. (2001). Prometheus-a compositional modeling tool for real-time systems.

[32] Basu, A., Bozga, M., & Sifakis, J. (2006, September). Modeling heterogeneous real-time components in BIP. In Software Engineering and Formal Methods, 2006. SEFM 2006. Fourth IEEE International Conference on (pp. 3-12). Ieee.

[33] Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A., & Sifakis, E. (2010, January). Verification of an afdx infrastructure using simulations and probabilities. In Runtime Verification (pp. 330-344). Springer Berlin Heidelberg.

[34] González Harbour, M., García, J. G., Gutiérrez, J. P., & Moyano, J. D. (2001). Mast: Modeling and analysis suite for real time applications. In Real-Time Systems, 13th Euromicro Conference on, 2001. (pp. 125-134). IEEE.

[35] Gutiérrez, J. J., Palencia, J. C., & Harbour, M. G. (2012). Response time analysis in AFDX networks with sub-virtual links and prioritized switches. XV Jornadas de Tiempo Real.

[36] García, J. G., Gutiérrez, J. P., & Harbour, M. G. (2000). Schedulability analysis of distributed hard real-time systems with multiple-event synchronization. InReal-Time Systems, 2000. Euromicro RTS 2000. 12th Euromicro Conference on (pp. 15-24). IEEE.

[37] Jiqiang, X., Weimin, Y., & Ronggang, B. (2011, July). Study on Real-Time Performance of AFDX Using OPNET. In Control, Automation and Systems Engineering (CASE), 2011 International Conference on (pp. 1-5). IEEE.

[38] TrueTime, http://www.control.lth.se/truetime/

[39] Li, J., Guan, H., Yao, J., Zhu, G., & Liu, X. (2012, November). Performance enhancement and optimized analysis of the worst case end-to-end delay for AFDX networks. In Green Computing and Communications (GreenCom), 2012 IEEE International Conference on (pp. 301-310). IEEE.

[40] Chen, X., Xiang, X., & Wan, J. (2009, June). A software implemetation of afdx end system. In New Trends in Information and Service Science, 2009. NISS'09. International Conference on (pp. 558-563). IEEE.

[41] Fernández, J., Pérez, H., Gutiérrez, J. J., & Harbour, M. G. (2015). AFDX Emulator for an ARINC-Based Training Platform. In Reliable Software Technologies–Ada-Europe 2015 (pp. 212-227). Springer International Publishing.

[42] Hornig, R.: Avionics Full-Duplex Switched Ethernet for OMNeT++ (2012).

https://github.com/omnetpp/afdx

[43] Khazali, I., Boulais, M. A., & Cole, P. (2009, October). AFDX software network stack implementation—Practical lessons learned. In Digital Avionics Systems Conference, 2009. DASC'09. IEEE/AIAA 28th (pp. 1-B). IEEE.

[44] Johnson, L. A. (1998). DO-178B, Software considerations in airborne systems and equipment certification. Crosstalk, October.

[45] Bril, R. J., Lukkien, J. J., & Verhaegh, W. F. (2009). Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. Real-Time Systems, 42(1-3), 63-119.

[46] Bril, R. J., Lukkien, J. J., & Verhaegh, W. F. (2007, July). Worst-case response time analysis of real-time tasks under fixed-priority scheduling with

deferred preemption revisited. In Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on (pp. 269-279). IEEE.

[47] Mäki-Turja, J., & Nolin, M. (2008). Efficient implementation of tight response-times for tasks with offsets. Real-Time Systems, 40(1), 77-116.

[48] Palencia, J. C., & Harbour, M. G. (1999). Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE (pp. 328-339). IEEE.

[49] Palencia, J. C., & Harbour, M. G. (2003, July). Offset-based response time analysis of distributed systems scheduled under EDF. In Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on (pp. 3-12). IEEE.

[50] Spuri, M. (1996). Holistic analysis for deadline scheduled real-time distributed systems.

[51] Tindell, K., & Clark, J. (1994). Holistic schedulability analysis for distributed hard real-time systems. Microprocessing and microprogramming, 40(2), 117-134.

[52] Rivas, J. M., Gutiérrez, J. J., Palencia, J. C., & Harbour, M. G. (2011, July). Schedulability analysis and optimization of heterogeneous edf and fp distributed real-time systems. In Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on (pp. 195-204). IEEE.

[53] Le Boudec, J. Y., & Thiran, P. (2001). Network calculus: a theory of deterministic queuing systems for the internet (Vol. 2050). Springer Science & Business Media.

[54] Keshav, S. (1997). Computer networking: An engineering approach.

[55] Okino, C. M. (1998). A framework for performance guarantees in communication networks (Doctoral dissertation, Ph. D. Dissertation, UCSD, 1998. 210 BIBLIOGRAPHY).

[56] Martin, S., & Minet, P. (2006, April). Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. In Parallel

and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International (pp. 8-pp). IEEE.

[57] Hua, Y., & Liu, X. (2011, April). Scheduling design and analysis for end-to-end heterogeneous flows in an avionics network. In INFOCOM, 2011 Proceedings IEEE (pp. 2417-2425). IEEE.

[58] Ji, X., Li, J., Li, H., Zhou, H., Hu, F., Liu, X., & Zhu, G. (2011, March). Analysis of Deterministic End-to-end Delay in Multi-hop AFDX Avionics Network System. In PECCS (pp. 434-440).

[59] Diez Barrero,D. (September 2009) Distributed avionics databus simulation. Individual research project MSc.Cranfield: Cranfield University, School of Engineering.

[60] Tommaso Falchi Delitala. (September 2009) Simulation of Switched Avionic Databus, A real-time software AFDX simulation. Individual research project MSc, Cranfield: Cranfield University, School of Engineering.

[61] Chen, T. (2011). Development and simulation of hard real-time switched-Ethernet avionics data network. Individual research project MSc, Cranfield: Cranfield University, School of Engineering.

[62] Rago, S. A. (1993). *UNIX System V network programming*. Pearson Education.

[63] Mauerer, W. (2010). *Professional Linux kernel architecture*. John Wiley & Sons.

[64] Student's T-test, https://en.wikipedia.org/wiki/Student%27s_t-test